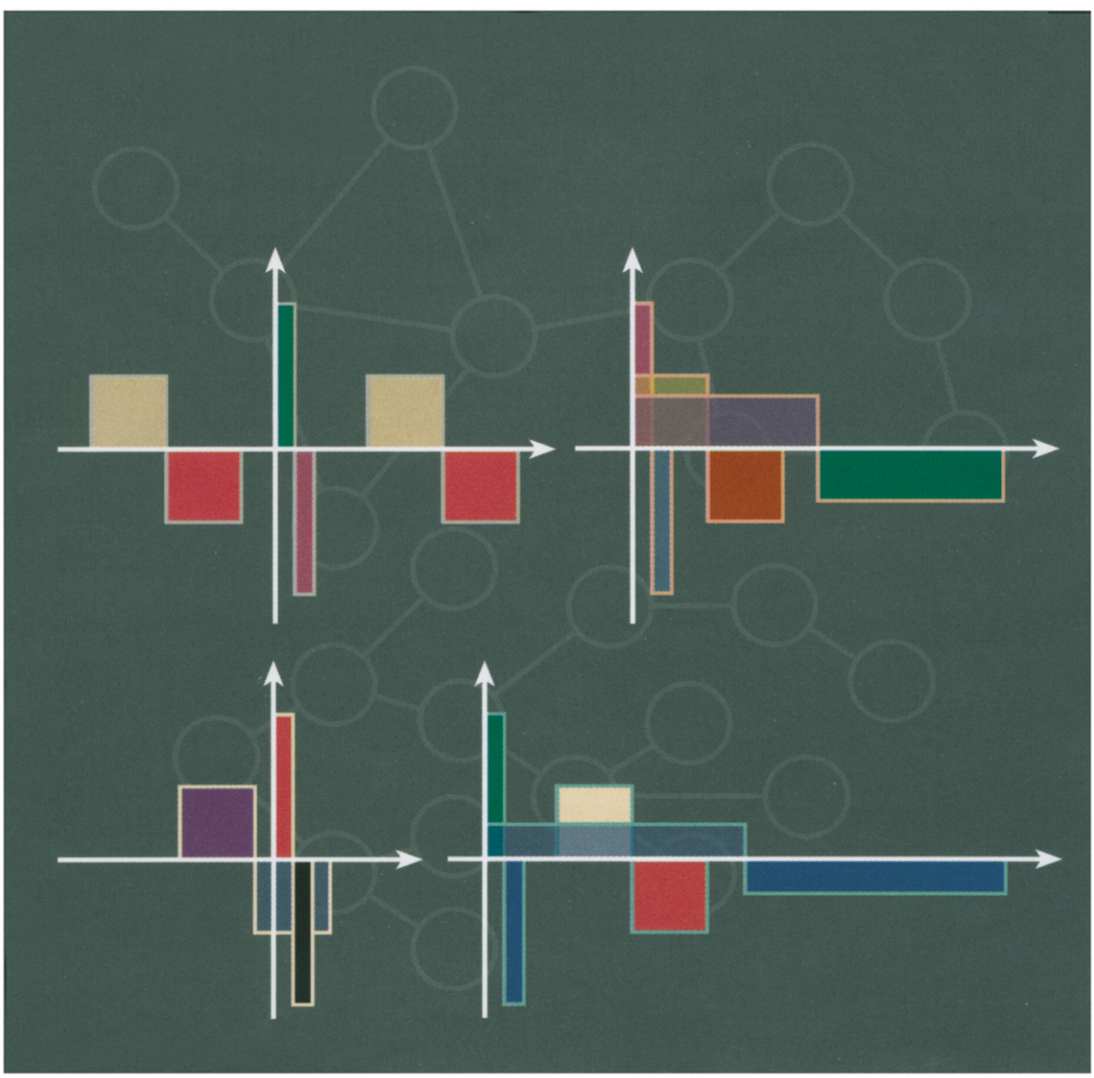
VISIT…

ELSEVIER
ACADEMIC
PRESS

MATHEMATICS *for* MULTIMEDIA

Mladen Victor Wickerhauser



Mathematics for Multimedia

This Page Intentionally Left Blank

Mathematics for Multimedia

Mladen Victor Wickerhauser
Washington University

2004

ACADEMIC PRESS

An imprint of Elsevier Science

AMSTERDAM – BOSTON – HEIDELBERG – LONDON – NEW YORK – OXFORD
PARIS – SAN DIEGO – SAN FRANCISCO – SINGAPORE – SYDNEY – TOKYO

Academic Press
An imprint of Elsevier Science
525 B Street, Suite 1900
San Diego, California 92101-4495
USA

© 2004 Elsevier Science (USA). All rights reserved.

This work is protected under copyright by Elsevier Science, and the following terms and conditions apply to its use:

Photocopying
Single photocopies of single chapters may be made for personal use as allowed by national copyright laws. Permission of the Publisher and payment of a fee is required for all other photocopying, including multiple or systematic copying, copying for advertising or promotional purposes, resale, and all forms of document delivery. Special rates are available for educational institutions that wish to make photocopies for non-profit educational classroom use.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, e-mail: permissions@elsevier.com. You may also complete your request on-line via the Elsevier Science homepage (http://www.elsevier.com), by selecting 'Customer Support' and then 'Obtaining Permissions'.

In the USA, users may clear permissions and make payments through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA; phone: (+1) (978) 7508400, fax: (+1) (978) 7504744, and in the UK through the Copyright Licensing Agency Rapid Clearance Service (CLARCS), 90 Tottenham Court Road, London W1P 0LP, UK; phone: (+44) 207 631 5555; fax: (+44) 207 631 5500. Other countries may have a local reprographic rights agency for payments.

Derivative Works
Tables of contents may be reproduced for internal circulation, but permission of Elsevier Science is required for external resale or distribution of such material.
Permission of the Publisher is required for all other derivative works, including compilations and translations.

Electronic Storage or Usage
Permission of the Publisher is required to store or use electronically any material contained in this work, including any chapter or part of a chapter.

Except as outlined above, no part of this work may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior written permission of the Publisher.
Address permissions requests to: Elsevier's Science & Technology Rights Department, at the phone, fax and e-mail addresses noted above.

Notice
No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein. Because of rapid advances in the medical sciences, in particular, independent verification of diagnoses and drug dosages should be made.

Library of Congress Cataloging in Publication Data
A catalog record from the Library of Congress has been applied for.

First edition 2004

ISBN: 0-12-748451-5

♾ The paper used in this publication meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of Paper).
Printed in Hungary.

For my daughter Natalie

This Page Intentionally Left Blank

Preface

Beautiful mathematical ideas abound in multimedia software! Some are not encountered until late in undergraduate or even postgraduate study, whereas they can be appreciated and used much earlier. This book presents a few dozen such pearls, strung together by their ubiquity in many applications. It is based on the course *Topics in Applied Mathematics: Multimedia Algorithms*, first taught at Washington University in the Fall of 1997.

The material is divided into six roughly equal chapters, intended for a pace of six lecture hours per chapter. Thus, the entire text may be covered in one semester, with time left for examinations and student projects as was done at Washington University. Alternatively, one or two of the last chapters may be omitted to fit the material into a trimester. There is a good deal of independence among the chapters, to permit such tailoring. Any single chapter, for example, can be a starting point for an undergraduate thesis in applied mathematics.

Algorithms are often divided into "integer" (discrete, exact) and "real" (continuous, approximate) types. Multimedia software is a big user of both types, combining them as needed, so this text does not enforce any kind of segregation. Understanding and analyzing a mix of procedures, with the assumptions, principles and techniques commonly used in their implementations, uncovers the mathematical gems in a more natural way than separation into "discrete mathematics" and "applied analysis."

In some cases, the mathematical theory is the main point. Proofs crucial to understanding are provided, at a level suited to mature undergraduates past Calculus. An appendix on basics is available to fill gaps. Further readings in more advanced sources are suggested at the end of each chapter. In other cases, the main point is practical implementation. Standards, conventions and common practice are discussed with reference to the actual standards documents. Key algorithms are presented in pseudocode and Standard C to assist programming, experimentation, and exercise solution. Some of the longer exercises are designed to give a taste of applied research, though complete solutions are supplied in the answers appendix. The shorter exercises can be easily modified for examinations.

I am grateful to my student, Dr. Wojciech Czaja, for his assistance in preparing some of the exercise solutions and for his numerous useful comments. I am also grateful to the National Science Foundation for the financial support, through grant DMS-9631359, that helped me create the course and this book.

St. Louis, Missouri
21 October 2002

This Page Intentionally Left Blank

Contents

1 Numbers and Arithmetic **1**
- 1.1 Integers . . . 2
  - 1.1.1 Modular arithmetic . . . 6
  - 1.1.2 Representing integers in binary computers . . . 8
  - 1.1.3 Integer arithmetic . . . 10
- 1.2 Real Numbers . . . 12
  - 1.2.1 Precision and accuracy . . . 14
  - 1.2.2 Representing real numbers . . . 15
  - 1.2.3 Propagation of error . . . 18
- 1.3 Exercises . . . 21
- 1.4 Further Reading . . . 22

2 Space and Linearity **23**
- 2.1 Vector Spaces . . . 23
  - 2.1.1 Euclidean space . . . 24
  - 2.1.2 Abstract vector spaces . . . 26
  - 2.1.3 Inner product spaces . . . 35
- 2.2 Linear Transformations . . . 42
  - 2.2.1 Matrix algebra . . . 43
  - 2.2.2 Adjoints and projections . . . 47
  - 2.2.3 Linear independence and invertibility . . . 53
  - 2.2.4 Solving linear systems of equations . . . 57
  - 2.2.5 Sparse matrices . . . 62
- 2.3 Exercises . . . 65
- 2.4 Further Reading . . . 68

3 Time and Frequency **69**
- 3.1 Fourier Analysis . . . 69
  - 3.1.1 Periodic functions . . . 70
  - 3.1.2 Localization . . . 72
  - 3.1.3 Fourier series . . . 83
- 3.2 Discrete Fourier Analysis . . . 94
  - 3.2.1 Discrete Fourier transform . . . 95

3.2.2 Discrete Hartley transform 98
3.2.3 Discrete sine and cosine transforms 100
3.3 Exercises 102
3.4 Further Reading 105

4 Sampling and Estimation **107**
4.1 Approximation and Sampling 108
4.1.1 Polynomial interpolation 108
4.1.2 Piecewise interpolation 115
4.1.3 Sampling spaces 117
4.2 Measurement and Estimation 123
4.2.1 Quantization, precision, and accuracy 123
4.2.2 Estimation 127
4.3 Exercises 130
4.4 Further Reading 132

5 Scale and Resolution **133**
5.1 Wavelet Analysis 133
5.1.1 Haar functions 134
5.1.2 The affine group 136
5.1.3 Wavelet transforms 140
5.2 Discrete Wavelet Transforms 144
5.2.1 Multiresolution analysis (MRA) 145
5.2.2 From MRAs to filters 148
5.2.3 From filters to discrete wavelet transforms 156
5.2.4 Lifting 166
5.3 Exercises 173
5.4 Further Reading 177

6 Redundancy and Information **179**
6.1 Information Source Coding 180
6.1.1 Lossless encoding 181
6.1.2 Efficient coding 186
6.1.3 Huffman's algorithm 191
6.2 Error Correction and Detection 197
6.2.1 Parity bits 197
6.2.2 Hamming codes 201
6.2.3 Checksums and cyclic redundancy codes 205
6.3 Exercises 218
6.4 Further Reading 220

A Answers **223**
A.1 ...to Chapter 1 Exercises 223
A.2 ...to Chapter 2 Exercises 229
A.3 ...to Chapter 3 Exercises 235

A.4 ...to Chapter 4 Exercises 244
A.5 ...to Chapter 5 Exercises 250
A.6 ...to Chapter 6 Exercises 263

B Basics, Technicalities, and Digressions **273**
B.1 ASCII and other character sets 273
B.2 Algorithms 275
B.3 Big-Oh notation 277
B.4 Calculus methods 278
B.4.1 Techniques of proof 279
B.4.2 Limits, continuity, and derivatives 280
B.4.3 Convergence of sequences, series and products 282
B.5 Some basic probability theory 288
B.6 Some more advanced results 292
B.7 Rising cut-off functions 294
B.8 Further Reading 295

This Page Intentionally Left Blank

Chapter 1

Numbers and Arithmetic

Processing, display, and communication of digital information, that is, information encoded as numbers, is accomplished by arithmetic with various kinds of numbers. Such computations are performed with *algorithms*, which are sequences of operations with numbers such as addition, multiplication, and reading and writing digits. Only finite algorithms can be used: these are procedures in which

- Every operation can be performed in a finite time;
- The algorithm is guaranteed to stop after a finite number of operations.

For an algorithm to be finite, its arithmetic operations can only be carried out to a finite degree of precision. In reality, a computer can keep only a small number of digits for each number because memory, processing and data communication are costly resources. But this usually poses no problems since the digital information of multimedia signals is itself of low precision. For example, a "CD-quality" digital sound recording consists of a sequence of numbers measuring the electrical output of a microphone at sequential times, with a precision of 5 decimal digits or less per measurement. Images from typical scanners are even less precise, consisting of arrays of numbers measuring light intensity to 3 decimal digits. Physical measurement is always imprecise, so these low precisions cannot be improved much. But the result is that computation for multimedia signal processing can be done with low precision arithmetic.

Most computers distinguish between *integers* and floating-point numbers or *floats*, which are approximations to real numbers. Either class is suitable for representing finite-precision information, but floats are somewhat more convenient for computation because the memory needed to store one of them is independent of its magnitude.

All computers have a fixed range of *representable values* for both integers and floats, and have efficient circuitry for arithmetic with numbers in those ranges. For example, most computers can perform floating-point arithmetic very efficiently at some built-in fixed precision, such as seven or 14 decimal places. More precision is

obtained with more complicated arithmetic algorithms that are consequently less efficient.

This chapter will discuss some of the mathematical properties of integer and floating-point arithmetic, define precision and analyze the propagation of error caused by low-precision computation, and describe a standard computer representation of floats with an explanation of how it aids efficient computation with good control of error propagation.

1.1 Integers

Denote the set of *integers* by $\mathbf{Z} \stackrel{\text{def}}{=} \{\ldots, -2, -1, 0, 1, 2, 3, \ldots\}$. The use of the letter 'Z' derives from the German noun "Zahl," or "number." Denote the *positive integers* by $\mathbf{Z}^+ \stackrel{\text{def}}{=} \{x \in \mathbf{Z} : x > 0\} = \{1, 2, 3, \ldots\}$ and the *negative integers* by $\mathbf{Z}^- \stackrel{\text{def}}{=} -\mathbf{Z}^+ = \{x \in \mathbf{Z} : x < 0\} = \{-1, -2, -3, \ldots\}$; neither subset contains zero, so $\mathbf{Z} = \mathbf{Z}^- \cup \{0\} \cup \mathbf{Z}^+$ is a disjoint union. Finally, denote the set of *natural numbers* by $\mathbf{N} \stackrel{\text{def}}{=} \mathbf{Z}^+ \cup \{0\} = \{x \in \mathbf{Z} : x \geq 0\} = \{0, 1, 2, \ldots\}$. An element of $\mathbf{Z}$ is determined by a finite list of symbols.

Division

For any integers a, b with $a \neq 0$, there are unique integers q and r, called the *quotient* and *remainder*, respectively, satisfying

1. $b = qa + r$;
2. $0 \leq r < |a|$.

If $a, b \in \mathbf{N}$, then $q, r \in \mathbf{N}$ as well. Quotient q is given by *integer division* $q = \left\lfloor \frac{b}{a} \right\rfloor$, where the *floor* function $\lfloor x \rfloor$ computes "the greatest integer less than or equal to x." Remainder r is the leftover: $r = b - qa$.

The "long division" algorithm can be used to determine q and r from a and b. Integer division takes finitely many steps: if a and b have at most N digits, then computing q and r requires $O(N^2)$ one-digit operations[1] such as trial multiplications or subtractions.

The Standard C programming language has special integer quotient and remainder operators / and % for computing q and r, as seen in this fragment of a computer program:

```
int a, b, q, r;
 ...
q = b/a;
r = b%a;
 ...
```

[1] See Section B.3 for an explanation of this "big-Oh" notation.

Standard C guarantees that conditions 1 and 2 hold for $a > 0$ and $b \geq 0$, but, unfortunately, condition 2 is not guaranteed if a or b is negative. For example, one typical machine computes as follows:

a	b	q=b/a	r=b%a
5	17	3	2
5	-17	-3	-2
-5	17	-3	2
-5	-17	3	-2

If $a \neq 0$ and the remainder upon division of b by a is zero, then a is said to *divide* b and is called a *divisor* of b, and b is said to be *divisible* by a. Some facts are:

- any a divides $b = 0$ (take $q = 0$ and $r = 0$ in $b = qa + r$);
- $a = 1$ divides any b (take $q = b$ and $r = 0$ in $b = qa + r$);
- if a divides b then $\pm a$ divides $\pm b$ (if $b = qa$, then $-b = -qa$, and so on);

Lemma 1.1 *If $b > 0$ and a divides b, then $0 < |a| \leq b$.*

Proof: Note that $b = |b| = |qa| = |q||a| > 0$, so $|q| \geq 1$. Thus $b - |a| = (|q| - 1)|a| \geq 0$, so $b \geq |a|$. Finally, $a \neq 0$ since $qa = b \neq 0$. □

Greatest common divisors

A positive integer c is said to be the *greatest common divisor* of two integers a and b if

gcd-1: c divides both a and b;

gcd-2: Any integer that divides both a and b also divides c.

If it exists, it must be unique by property gcd-2 and the last remark of the previous section. The proof is that if c_1 and c_2 are both greatest common divisors of a and b, then c_1 divides c_2 so $c_1 \leq c_2$, but then also c_2 divides c_1 so $c_2 \leq c_1$. Thus $c_2 = c_1$. But existence is guaranteed, too:

Theorem 1.2 *Every pair of integers a, b, not both zero, has a greatest common divisor, which can be written as $xa + yb$ for some integers x, y.*

Proof: Let $D = \{xa + yb : x, y \in \mathbf{Z}\}$. D contains some nonzero integers since not both a and b are zero, so D must contain some positive integers since $d \in D \Rightarrow -d \in D$. Let $c = x_0 a + y_0 b$ be the smallest positive integer in D. Then any integer z that divides both a and b will divide c, since $a = nz$ and $b = mz$ implies $c = (x_0 n + y_0 m)z$. Hence c satisfies property gcd-2.

To show that c divides a, write $a = qc + r$ with $0 \leq r < c$. Then $a = q(x_0 a + y_0 b) + r$, so $r = (1 - qx_0)a + (-qy_0)b \in D$. Then r must be zero, since otherwise it

would be a smaller positive element of D than c. The same argument shows that c divides b, so c satisfies property gcd-1. □

We write $c = \gcd(a, b)$. For example, $\gcd(-12, 16) = 4$. Note that $\gcd(0, 0)$ is undefined since every integer, no matter how large, divides both zeroes. Hence, the "not both zero" is a necessary assumption.

By convention, $\gcd(a, 0) = |a|$ for any $a \neq 0$. Other useful facts are:

- $\gcd(a, b) = \gcd(b, a) = \gcd(|a|, |b|)$.
- If $a' = \max(|a|, |b|)$ and $b' = \min(|a|, |b|)$, then $\gcd(a, b) = \gcd(a', b')$.
- $\gcd(a, b) \leq \min(|a|, |b|)$.
- If a divides b, then $\gcd(a, b) = |a|$.

An efficient algorithm for computing greatest common divisors has been known for thousands of years, and was written down by Euclid around 300 BC. To start it off, first prepare the inputs by replacing $a \leftarrow \max(|a|, |b|)$ and $b \leftarrow \min(|a|, |b|)$, so as to guarantee that $a > 0$, $b \geq 0$, and $a \geq b$:

Euclid's Algorithm

```
gcd( a, b ):
[0] Let c = a
[1] Let a = b%a
[2] Let b = c
[3] If a>0, then go to [0]
[4] Print b
```

To analyze this algorithm, let a_n, b_n be the respective values of a, b after the n^{th} visit to step 2. Step 1 insures that $a > a_1 > a_2 > \cdots \geq 0$, and since each a_n is an integer, the loop must terminate after at most a steps. Steps 0 and 2 require copying the digits of a and c, step 1 is the division algorithm, step 3 requires reading the digits of a number to see if they are all 0, and step 4 requires printing the digits of a number. Hence, each step takes finitely many calculations, so the algorithm is finite.

Suppose $k \geq 1$ is the least index for which $a_k = 0$. Then a_{k-1} divides b_{k-1}, so the printed value is $b_k = a_{k-1} = \gcd(a_{k-1}, b_{k-1})$.

Note that any common divisor of both a and b also divides both $a_1 = b\%a$ and $b_1 = a$. For $n = 1, 2, \ldots, k$, the same observation reveals that any common divisor of a_n and b_n is a common divisor of both a_{n+1} and b_{n+1}. Hence, by induction on n, the set of common divisors of a and b equals the set of common divisors of a_n and b_n. In particular, these sets have the same largest element, $\gcd(a_n, b_n) = \gcd(a, b)$ for all $1 \leq n < k$, and the printed value will be $\gcd(a_{k-1}, b_{k-1}) = \gcd(a, b)$.

How many iterations through steps 0–2 will Euclid's algorithm take? Recall that for $1 \leq n < k$, $a_{n+1} < a_n$, so $a_{n+1} = a_n - d_n$ for some $0 < d_n \leq a_n$. But also, $b_{n+1} = a_n$, so for any $1 \leq n \leq k - 2$, $a_{n+2} = b_{n+1}\%a_{n+1} = a_n\%a_{n+1}$, which implies two things: $a_{n+2} < a_{n+1} = a_n - d_n$, and also $a_{n+2} = a_n\%(a_n - d_n) \leq d_n$.

Thus $a_{n+2} \leq \min\{a_n - d_n, d_n\} \leq \frac{1}{2}a_n$. Thus the number of iterations required by Euclid's algorithm is at most $2\log_2 a$, which is $O(N)$ for N-digit inputs a.

Primes

Integers a and b are called *relatively prime* if $\gcd(a,b) = 1$. Any integer a is relatively prime to $b = 1$.

Lemma 1.3 *If c divides ab and $\gcd(a,c) = 1$, then c divides b.*

Proof: Write $1 = \gcd(a,c) = m_0 a + n_0 c$ as in the proof of Theorem 1.2. Then $b = m_0 ab + n_0 cb$. Since c evidently divides $n_0 cb$, and c divides $m_0 ab$ by assumption, it follows that c divides $m_0 ab + n_0 cb = b$. □

An integer $p > 1$ is called *prime* if its only divisors are ± 1 and $\pm p$. Thus, if a is any integer and p is prime, either p divides a or else a and p are relatively prime. It follows from Lemma 1.3 that if p is prime and p divides ab, then either p divides a or p divides b.

See if you can prove from the above definitions that distinct primes are relatively prime.

There are infinitely many prime numbers since any finite list $p_1, p_2, \ldots, p_n$ must omit a prime divisor of $1 + p_1 p_2 \cdots p_n$. The smallest primes are 2, 3, 5, 7, 11, 13, and 17.

Unique factorization

Suppose that $N > 1$ is a fixed integer. Then either N is prime or there is some $1 < a < N$ that divides N. The same argument may be repeated for $N' = a$ and $N'' = N/a$, both of which are strictly less than N. Thus, finitely many steps will yield a *prime factorization* $N = p_1 p_2 \cdots p_k$.

Theorem 1.4 *Prime factorization is unique: If $p_1 \cdots p_n = q_1 \cdots q_m$ and the p's and q's are primes, then $n = m$ and, possibly after re-indexing, the p's are the same as the q's.*

Proof: Let r be one of the primes $\{q_1, q_2, \ldots, q_m\}$. Since r divides $p_1 \cdots p_n$, it must divide one of the p's. But then r must equal one of the p's since two primes are either equal or relatively prime. Thus the set of p's includes all the q's. Similarly, the set of q's includes all the p's. If a prime r appears i times in one factorization and $j > i$ times in the other, then dividing both by r^i leaves equal factorizations with $j - i > 0$ factors r in one but no factor r in the other, which is not possible. Thus the count of each prime must be the same in both factorizations. □

Unique factorization requires that 1 not be considered prime. Computing the prime factorization of a large integer cannot be done fast by any known method, and this tough problem can be used for *cryptography*.

1.1.1 Modular arithmetic

Fix an integer $M > 1$ and say that two integers a and b are *congruent modulo* M if M divides $b - a$, that is, if a and b differ by a multiple of M. Such a condition is written $a \equiv b \pmod{M}$.

Congruent numbers must leave the same remainders $a\%M$ and $b\%M$, so the finite set $\{n : 0 \leq n < M\} = \{0, 1, \ldots, M-1\}$, which may also be called M, contains one representative from each of the *congruence classes* modulo M. Namely, every integer is congruent to one of the numbers $0, 1, \ldots$, or $M - 1$, modulo M.

Modular addition, subtraction, and multiplication is similar to ordinary arithmetic except that equality is replaced by congruence. Thus the answer need only be determined within an integer multiple of the modulus M, and the operands can be replaced by congruent representatives from the set M:

Lemma 1.5 *If $a, b, c \in \mathbf{Z}$ are respectively congruent to $\alpha, \beta, \gamma \in M$ modulo M, then*

$$ab + c \equiv \alpha\beta + \gamma \pmod{M}.$$

Proof: Write $a = \alpha + xM$, $b = \beta + yM$, and $c = \gamma + zM$, where x, y, and z are integers. Then $ab + c = \alpha\beta + \gamma + (\alpha y + \beta x + z)M$, so $ab + c - (\alpha\beta + \gamma)$ is an integer multiple of the modulus M. □

The modular additive inverse of x is any number y such that $x + y \equiv 0 \pmod{M}$. For $x \in M$, a natural candidate is $y = M - x$, which also belongs[2] to the set M. Thus we can mimic ordinary signed integers in modular arithmetic by considering numbers between 0 and $M/2$ to be positive, and those between $M/2$ and M to be negative. Modular addition and subtraction will then agree with ordinary addition and subtraction for all operations with integers of sufficiently small magnitude. If $|x| < M/4$ and $|y| < M/4$, then $x + y$ will have the same representative in $\mathbf{Z}$ as in the signed interpretation of the set M.

Modular multiplication likewise agrees with ordinary integer multiplication of small enough integers. Where $M/4$ was a magnitude limit for addition, it is $\sqrt{M/2}$ for multiplication.

Modular division b/a can sometimes be done even if a does not divide b. For example, $5 \cdot 2 \equiv 1 \pmod{9}$, so we can write $1/2 \equiv 5 \pmod{9}$ or $1/5 \equiv 2 \pmod{9}$. Define a *quasi-inverse* of a modulo M to be any integer a' satisfying

$$aa' \equiv 1 \pmod{M}. \tag{1.1}$$

This is a multiplicative inverse in modular arithmetic, but it has no analog in ordinary integer arithmetic.

Lemma 1.6 *Let $M > 1$ and a be integers. Then a has a quasi-inverse a' modulo M if and only if $\gcd(a, M) = 1$, and in that case this a' is unique in the set $\{1, \ldots, M-1\}$.*

[2] What about $x = 0$?

Proof: If $\gcd(a, M) = 1$, then write $1 = \gcd(a, M) = m_0 a + n_0 M$ as in the proof of Theorem 1.2. Evidently m_0 is a quasi-inverse of a, and if $m_0 \geq M$ or $m_0 < 0$, an appropriate multiple of M can be added to get a quasi-inverse $a' = m_0 + kM \in \{0, 1, 2, \ldots, M-1\}$. But this a' cannot be zero since $a \cdot 0 \equiv 0 \not\equiv 1 \pmod{M}$. For uniqueness, note that if both a' and a'' satisfy $aa' \equiv aa'' \equiv 1 \pmod{M}$, then M divides $a(a' - a'')$. By Lemma 1.3, M must divide $a' - a''$, so if both a' and a'' lie in the set $\{1, \ldots, M-1\}$ they must be equal.

On the other hand, if $\gcd(a, M) = m_0 a + n_0 M = d > 1$, then there are no integers x, y such that $0 < ax + My < d$. In particular, there are none that give $ax = 1 + yM$, so there is no integer x solving $ax \equiv 1 \pmod{M}$. □

Corollary 1.7 *If M is prime, then every integer a in the set $\{1, \ldots, M-1\}$ has a quasi-inverse modulo M.* □

The number of integers in $\{1, \ldots, M-1\}$ which are relatively prime to M, and which are therefore quasi-invertible modulo M, defines *Euler's function* $\phi = \phi(M)$. We see that $\phi(p) = p - 1$ for any prime number p, and in general:

Theorem 1.8 *Given the prime factorization $M = p_1^{m_1} \cdots p_n^{m_n}$, where $\{p_k : k = 1, \ldots, n\}$ are distinct primes and $\{m_k : k = 1, \ldots, n\}$ are positive integers, we have*

$$\phi(M) \stackrel{\text{def}}{=} \#\{k \in M : \gcd(k, M) = 1\} = \prod_{i=1}^{n} \left(p_i^{m_i} - p_i^{m_i - 1}\right),$$

where the symbol $\prod$ denotes the product of the terms that follow.

Proof: First, compute $\phi(p^m) = p^m - p^{m-1}$ for any individual prime p and any positive integer power m since the only numbers in $\{0, 1, 2, \ldots, p^m - 1\}$ which are not relatively prime to p^m are the multiples of p: $0p, 1p, \ldots, (p^{m-1} - 1)p = p^m - p$, of which there are evidently p^{m-1}.

Next, note that if $\gcd(M, N) = 1$, then $\phi(MN) = \phi(M)\phi(N)$. To prove this, write $\gcd(M, N) = 1 = x_0 M + y_0 N$ by Theorem 1.2 and observe that any integer k can be written as $k = kx_0 M + ky_0 N = xM + yN$ for some $x, y \in \mathbf{Z}$. On the other hand, the integers MN and $k = xM + yN$ are relatively prime if and only if $\gcd(x, N) = 1$ and $\gcd(M, y) = 1$. Hence $\{k \in \mathbf{Z} : \gcd(k, MN) = 1\}$ is the set

$$\{xM + yN : \ x, y \in \mathbf{Z};\ \gcd(x, N) = 1;\ \gcd(M, y) = 1\}.$$

But $xM + yN \equiv x'M + y'N \pmod{MN}$ if and only if $(x - x')M = (y' - y)N + kMN$ for some integer k, which is true if and only if N divides $(x - x')$ and M divides $(y' - y)$, so $xM + yN \equiv x'M + y'N \pmod{MN}$ if and only if $x \equiv x' \pmod{N}$ and $y \equiv y' \pmod{M}$.

Thus each integer in $\{0, 1, \ldots, MN\}$ that is relatively prime with MN is realized as $xM + yN \pmod{MN}$ for exactly one of the $\phi(N)$ representatives $x \in \{0, 1, \ldots, N\}$ and exactly one of the $\phi(M)$ representatives $y \in \{0, 1, \ldots, M\}$ that are relatively prime to N and M, respectively. This implies that $\phi(MN) = \phi(M)\phi(N)$.

Finally, since powers of distinct primes are relatively prime, we can factor ϕ to get the result by our first observation: $\phi(\prod_i p_i^{m_i}) = \prod_i \phi(p_i^{m_i}) = \prod_i \left(p_i^{m_i} - p_i^{m_i-1}\right)$. □

The following extension of Euclid's algorithm, from Knuth's *Fundamental Algorithms*, page 14, finds quasi-inverses. Given two positive integers a and b, it computes $d = \gcd(a, b)$ and two integers x, y satisfying $ax + by = d$:

Extended Euclid's Algorithm

```
gcdx( a, b ):
[0] Initialize x=0, y=1, xo=1, yo=0, c=a, and d=b
[1] Let q = c/d and r = c%d
[2] If r==0, then go to [5]
[3] Let c = d, d = r, t = xo, xo = x, x = t-q*x,
        t = yo, yo = y, and y = t-q*y
[4] Go to [1]
[5] Print x, y, and d
```

Starting with relatively prime a and $b = M$, the output will be a quasi-inverse x of a, some integer y, and the known result $d = \gcd(a, M) = 1$. After storing $k = \lfloor x/M \rfloor$, we adjust $x \leftarrow x - kM$ to get a quasi-inverse in $\{0, 1, \ldots, M-1\}$. Note that this requires adjusting $y \leftarrow y + ka$ to preserve the equality $ax + by = d$.

1.1.2 Representing integers in binary computers

Computers have *internal representations* for numbers that in most cases are easily translated to binary, or base-2, notation. Most humans, on the other hand, use decimal or base-10 notation. Binary notation in this text will be indicated by a string of binary digits, or *bits*, each taking the value 0 or 1, followed by "base 2" in parentheses. A four-bit binary number will look like $b_3b_2b_1b_0$ (base 2); one specific example is the number 9, which is 1001 (base 2). This is analogous to a four-digit decimal number like $d_3d_2d_1d_0$ (base 10), for example 1997 (base 10). The "base 10" is usually omitted.

The base can be any positive integer greater than one.[3] A number can be written in any base, and its value for use in arithmetic can be obtained by summing the values represented by its digits:

$$\begin{aligned} 1001 \text{ (base 2)} &= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 9 \text{ (base 10)}; \\ 1997 \text{ (base 10)} &= 1 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 + 7 \times 10^0 \\ &= 11111001101 \text{ (base 2)}. \end{aligned}$$

More generally, the n-digit number written as $h_{n-1} \ldots h_1 h_0$ (base B) has the value $h_{n-1} \times B^{n-1} + \cdots + h_1 \times B^1 + h_0 \times B^0$. The digits $h_0, h_1, \ldots$ must lie in the range

[3] We will not consider tallies, like 3 = ||| or 7 = ||||| ||, that give "base one" notation.

Hex	Bin	Hex	Bin	Hex	Bin	Hex	Bin
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Table 1.1: One hexadecimal digit corresponds to four binary digits, or bits.

$\{0, 1, \ldots, B-1\}$. The choice $B = 16$ gives the useful *hexadecimal* system which uses the digits $\{0, \ldots, 9, A, B, C, D, E, F\}$, where $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, and $F = 15$. Hexadecimal and binary are related by Table 1.1. Each hexadecimal digit corresponds to 4 bits, making it easy to find the corresponding binary expression: 1997 (base 10) = $7CD$ (base 16) = 0111 1100 1101 (base 2).

The binary digits $b_0, b_1, b_2, \ldots$ of a nonnegative integer x can be generated by the following algorithm:

Compute the Binary Digits of an Integer $x \geq 0$

```
bits( x ):
[0] Print x%2
[1] Let x = x/2
[2] If x>0, then go to [0]
```

Notice that this prints bits in reverse order. For positive x, it first prints the *least significant bit* b_0, then b_1, b_2, and so on, terminating with the *most significant bit*, the leftmost "1" of x in binary. If $x = 0$, this routine prints a single 0.

The digits in base B of a nonnegative integer x are generated by a similar algorithm. Recall that, if $x \geq 0$ and $B > 0$ are integers, then $x\%B$ is the remainder left after dividing x by B.

Compute the Base-B Digits of an Integer $x \geq 0$

```
digits( x, B ):
[0] Print x%B
[1] Let x = x/B
[2] If x>0, then go to [0]
```

Of course, if $B > 10$, then a suitable letter should be printed to represent digit values from 10 to $B - 1$.

Fractions and "decimals" can also be written in any base, using the following interpretation: evaluate $x = h_{n-1} \ldots h_1 h_0 . h_{-1} h_{-2} \ldots h_{-m}$ (base B) as

$$x = h_{n-1} \times B^{n-1} + \cdots + h_1 \times B + h_0 + \frac{h_{-1}}{B} + \frac{h_{-2}}{B^2} + \cdots \frac{h_{-m}}{B^m}. \tag{1.2}$$

The "decimal" point separates the integer and fractional parts of the number. In

this example, the fractional part of x can be rewritten as

$$\frac{h_{-1} \times B^{m-1} + h_{-2} \times B^{m-2} + \cdots + h_{-m}}{B^m}. \tag{1.3}$$

The integer numerator from Equation 1.3, which is B^m times the fractional part of x, may be used as input to the base-B conversion program. Its output will be the m digits to the right of the decimal point, printed in reverse order.

1.1.3 Integer arithmetic

A binary computer that stores w bits per integer has a maximum unsigned integer of $2^w - 1$. In general, a computer that stores integers as w base-B digits has a maximum unsigned integer of $B^w - 1$. However, the case $B \neq 2$ is rare enough to be ignored hereafter. Some common values for w are 8, 16, 24, 32, 36, 64, 80, 96, or 128, typically with a selection of several being available. For example, a program written in the Standard C language on one machine *host* can use integer variables of type `char` ($w = 8$), `short` ($w = 16$), `int` ($w = 32$), or `long` ($w = 64$). These parameters are stored in a file named `limits.h` on each machine:

Excerpt from `limits.h`

```
#define    CHAR_BIT                    8
#define    SCHAR_MIN                -128
#define    SCHAR_MAX                 127
#define    UCHAR_MAX                 255
#define    SHRT_MIN                -32768
#define    SHRT_MAX                 32767
#define    USHRT_MAX                65535
#define    INT_MIN             -2147483648
#define    INT_MAX              2147483647
#define    UINT_MAX             4294967295
#define    LONG_MIN    -9223372036854775808
#define    LONG_MAX     9223372036854775807
#define    ULONG_MAX   18446744073709551615
```

In particular, 32-bit binary integers of type `unsigned int` can take one of the $2^{32} = 4\,294\,967\,296$ possible values between 0 and $2^{32} - 1 = 4\,294\,967\,295$. The special name `UINT_MAX` is given to this *maximum unsigned integer*, or largest counting number.

If x and y are positive integers with $x + y >$ `UINT_MAX`, the addition $x + y$ cannot be performed using variables of type `unsigned int`. However, the sum will be correctly computed on any machine with a maximum unsigned integer of $x + y$ or greater, so it may be possible to perform the calculation with variables of type `long int` or `unsigned long int`. The 1999 C standard includes the types `long long int` and `unsigned long long int`, too, which may have even more bits on a particular host.

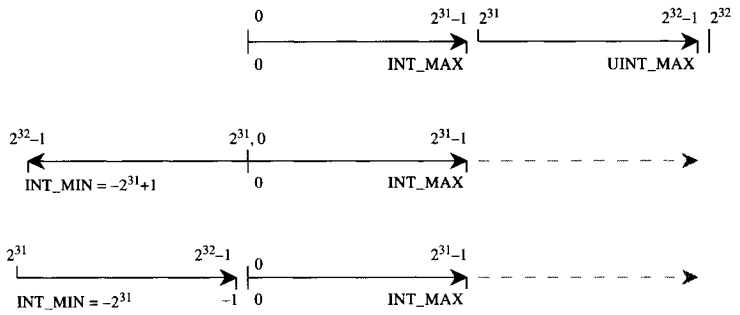


Figure 1.1: Top: Unsigned 32-bit integers. Middle: Sign and magnitude interpretation. Bottom: Twos complement interpretation. Numbers above each line are the counting values, while those below the line are the represented values.

There are two common ways of representing negative integers at fixed binary precision. The first is called *sign and magnitude form*, and consists of treating the most significant bit as a sign bit. If 0, the number is positive. If 1, the number is negative. The remaining bits are taken to be the absolute value of the integer and are interpreted as counting numbers. This method wastes one binary string representing -0. To change $x \mapsto -x$, just change the sign bit to its complement. The most negative 32-bit integer representable by sign and magnitude is $-(2^{31}-1) = -2\,147\,483\,647$, and the largest positive signed integer is $2^{31}-1 = 2\,147\,483\,647$ in 32-bit ones-complement arithmetic. Standard C defines `INT_MIN` and `INT_MAX`, respectively, to have these values on each host computer employing sign and magnitude representation.

The second, more common integer representation is called *twos complement form*. In this method, using w bits for a total of $2^w - 1$ numbers, the low half $[0, 2^{w-1}-1]$ represent nonnegative integers, while the high half $[2^{w-1}, 2^w - 1]$ represent negative integers to which 2^w has been added. The most significant bit again determines the sign: 1 means negative, 0 means positive. This arrangement is depicted in Figure 1.1. It in turn has the drawback that the negative of a representable integer is not always representable since `-INT_MIN` is larger than `INT_MAX`.

The twos complement form of $-x$ for a w-bit integer x is the counting number represented by $2^w - x$, that is, the additive inverse of x modulo 2^w. It is therefore a simple bitwise operation to compute $x \mapsto -x$ in twos complement form: first find the *ones-complement* by flipping $0 \leftrightarrow 1$ all the bits of x, and then increment $x \leftarrow x+1$. Flipping, or *complementing* a w-bit number x is the same as subtracting it from $2^w - 1$, which is a string of w 1-bits, so these operations give $[(2^w - 1) - x] + 1 = 2^w - x$. For example, with $w = 8$ and $x = 13 = 00001101$ (base 2), the ones complement of x is $242 = 11110010$ (base 2), and the twos complement is $243 = 11110011$ (base 2), which is congruent to $-13 \pmod{256}$.

Twos complement w-bit integer arithmetic is implemented in hardware as arithmetic modulo 2^w, with addition, subtraction, multiplication, integer division and

remainder performed by dedicated circuitry. Some checks must be built in, though. It is possible to add two positive integers and get the representation of a negative integer, for example $100 + 99$ gives the 8-bit twos complement representation for -57. This is called an *integer overflow*. Likewise, $-100 - 99$ produces the *integer underflow* value 57. We can show that underflow or overflow occurs if and only if the carry into the sign or most significant bit is different from the carry out of the sign bit.

Logical operations such as order comparison and equality testing can be implemented by subtraction modulo 2^w followed by testing the sign bit or testing if all bits are zero.

1.2 Real Numbers

Denote by $\mathbf{Q}$ the set of *rational numbers* p/q, each of which is described[4] by a numerator $p \in \mathbf{Z}$ and a denominator $q \in \mathbf{Z}$, $q \neq 0$. Each element of $\mathbf{Q}$ is therefore completely described by a finite list of symbols. Of course, $p/q = p'/q'$ if and only if $pq' = p'q$, but we can always find the unique representative of p/q in *lowest terms* by the reduction $p \leftarrow p/\gcd(p,q)$, $q \leftarrow q/\gcd(p,q)$, followed by changing the sign of both p and q, if necessary, to make $q > 0$. A computer can store one rational number in the same space needed for one integer, just by assigning a fixed subset of bits for a nonnegative integer numerator, one bit for the sign, and the remaining bits for the positive integer denominator.

Devices to perform rational number arithmetic are combinations of devices that perform integer arithmetic. For example, $p/q + p'/q' = (pq' + p'q)/qq'$ requires three integer multiplications and one addition, with the signs carried by the numerators and the denominators taken to be positive. Likewise, comparisons are derived from integer comparisons: $p/q < p'/q'$ if and only if $pq' < p'q$, and so on.

It has been known since ancient times that fairly simple problems have no solutions in the rational numbers. A famous example, due to Euclid, is that $\sqrt{2}$ cannot be expressed as p/q for integers p, q since then $p^2/q^2 = 2$ implies that p^2 is even, so p must be even, and then p^2 is really divisible by 4 so q^2 and thus q must be even. Hence p/q is not in lowest terms. But this applies to every $p/q = \sqrt{2}$, so were $\sqrt{2}$ rational, it would have no representative in lowest terms, which cannot be. Hence we cannot solve the problem $x^2 = 2$ with $x \in \mathbf{Q}$.

We can easily find approximate solutions $x \in \mathbf{Q}$ to the problem $x^2 = 2$:

$$\begin{aligned}
1^2 &\leq 2 \leq 2^2 && \Rightarrow 1 \leq \sqrt{2} \leq 2;\\
1.4^2 &\leq 2 \leq 1.5^2 && \Rightarrow 1.4 \leq \sqrt{2} \leq 1.5;\\
1.41^2 &\leq 2 \leq 1.42^2 && \Rightarrow 1.41 \leq \sqrt{2} \leq 1.42;\\
1.414^2 &\leq 2 \leq 1.415^2 && \Rightarrow 1.414 \leq \sqrt{2} \leq 1.415;\\
&\vdots
\end{aligned}$$

[4] We write p/q for convenience, we do not actually divide.

$$1.4142135623^2 \leq 2 \leq 1.4142135624^2$$
$$\vdots \qquad \Rightarrow 1.4142135623 \leq \sqrt{2} \leq 1.4142135624;$$

This procedure can be continued indefinitely, with each step taking a finite amount of work to shrink the difference between the upper and lower estimate by a factor of 10.

A *Cauchy sequence* is an unending list $\{x_1, x_2, \ldots\}$ of numbers with the property that for every positive integer d, there some start index $N = N(d)$ such that $x_N, x_{N+1}, \ldots$ all have the same first d digits in their decimal expansion. For example, the rational approximations to $\sqrt{2}$ given by the lower bounds $\{1, 1.4, 1.41, \ldots\}$ define a Cauchy sequence with $N(d) = 10^d$ since the first d digits of x_n will be the same for all $n \geq N(d)$. It is an easy exercise to show that the following, more traditional definition is equivalent:

Definition 1 *The list* $\{x_1, x_2, \ldots\}$ *is a* Cauchy sequence *if, for every* $\epsilon > 0$, *there is some* $N = N(\epsilon)$ *such that* $|x_n - x_m| < \epsilon$ *whenever both* $n \geq N$ *and* $m \geq N$.

We may define the *real numbers* $\mathbf{R}$ to be the enlargement of $\mathbf{Q}$ that includes all the *infinite decimal expansions* obtained using Cauchy sequences of rational numbers. Since rational numbers themselves have decimal expansions, this means that $\mathbf{R}$ is the set of infinite decimal expansions. Numbers like 1 can also be considered infinite decimals $1.00\ldots$, and so on.

Not only are there infinitely many real numbers, but most of them cannot be specified exactly since that would require writing infinitely many digits. However, a fixed-precision approximation to a real number often suffices. Humans write such approximate values in *scientific notation* using base 10. For example, we write $.314159 \times 10^1$ for the six-digit approximation to π. Computers unable to print superscripts would write `.314159e+01`. The *mantissa* or *fractional part* `314159` contains the six digits, while the *exponent* `+01` indicates how to move the decimal point, here one digit to the right, in order to get the usual decimal expansion 3.14159.

Note that $1 = 0.999\ldots$ since $x = 0.99\ldots$ satisfies $10x - x = 9$, so two different decimal expansions can represent the same real number. We will say that two numbers $x, y \in \mathbf{R}$ are equal if, for any Cauchy sequences $\{x_n\}$ and $\{y_n\}$ representing x and y, respectively, the Cauchy sequence $\{x_n - y_n\}$ represents 0.

The real number x represented by a Cauchy sequence $\{x_n : n = 1, 2, \ldots\}$ is called its *limit*, and we write $x = \lim_{n\to\infty} x_n$. For every nonnegative integer d, there is an integer $N = N(d)$ such that the first d digits of $x_N, x_{N+1}, \ldots$ all agree with the first d digits of x. This is equivalent to the traditional definition:

Definition 2 *We say that the limit* $x = \lim_{n\to\infty} x_n$ *exists if, for every* $\epsilon > 0$, *there is some* $N = N(\epsilon)$ *such that* $|x - x_n| < \epsilon$ *whenever* $n \geq N$.

If the limit of some sequence $\{x_n\}$ exists, we also say that x_n *converges to* x *as* n *tends to infinity*, and write $x_n \to x$ as $n \to \infty$. By our construction, every Cauchy sequence of rational numbers has a unique real number as a limit. But once we include those limits, we have a complete set:

Theorem 1.9 (Completeness of R) *A Cauchy sequence of real numbers has a unique real-number limit.*

Proof: Suppose $x_1, x_2, \ldots$ is a Cauchy sequence of real numbers. For each $d = 1, 2, \ldots$, let $N = N(d)$ be an integer such that the first d digits of $x_N, x_{N+1}, \ldots$ are the same. The first d digits of x_N define a rational number[5], which we may call x'_d. But then $\{x'_1, x'_2, \ldots\}$ is a Cauchy sequence of rational numbers that converges to a unique real number, which we may denote by x. But x_n converges to x too, since for every $\epsilon > 0$ there is some d with $10^{-d} < \epsilon$, and $|x_n - x| < \epsilon$ for all $n > N(d)$. □

1.2.1 Precision and accuracy

Accuracy is the difference between the exact value of a quantity and its approximate value. *Precision* is the difference between two adjacent approximate values. To say that π is approximately 3.140000000 is to be precise but inaccurate. Using great precision for quantities known to low accuracy is misleading.

Suppose that $x_0 \neq 0$ is a real number, considered to be the exact value of a quantity, and x is another real number used to approximate x_0. For example, x might be an integer, or one of the representable real numbers available in 32-bit IEEE floating-point format. Then:

- The *absolute error* of approximating x_0 by x is $|x - x_0|$, also written $|\Delta x|$;
- The *relative error* of approximating x_0 by x is $|x - x_0|/|x_0|$, also written $|\Delta x|/|x_0|$;
- The number of *digits of accuracy* in x is the largest natural number d such that $10^d|\Delta x|/|x_0| \leq 5$. If the relative error is smaller than 1, then

$$d = \left\lfloor \log_{10}\left(\frac{5|x_0|}{|\Delta x|}\right)\right\rfloor .$$

- The number of *bits of accuracy* in an approximation of $x_0 \neq 0$ is the largest natural number b such that $2^b|\Delta x|/|x_0| \leq 1$. Again, if the relative error is smaller than 1, then

$$b = \left\lfloor \log_2\left(\frac{|x_0|}{|\Delta x|}\right)\right\rfloor .$$

Bits and digits of accuracy are related; each digit of accuracy is worth $\log_2(10) \approx 3.32$ bits, so $b \approx 3.32d$.

When $x_0 = 0$, absolute error is still $|\Delta x|$, but relative error is undefined and digits of accuracy is calculated using absolute error.

To say that 3.14159 is the six-digit approximation to π means that $3.14158 < \pi < 3.14160$. The absolute error $|\pi - 3.14159|$ is also called the *round-off error*, or sometimes the *truncation error*. It is always smaller than one unit at the least

[5] What power of 10 will be the denominator?

significant digit of the mantissa. The *error interval* is determined by how the approximate value was chosen. It is $3.14159 \leq \pi < 3.14160$ if the approximation just truncated a longer decimal expansion, but it is $3.141585 \leq \pi \leq 3.141595$ if the approximation rounded to the nearest six-digit decimal expansion.

1.2.2 Representing real numbers

A computer can only distinguish among finitely many *representable values*, lying in some bounded range, and the format in which this is usually done is called *floating-point*. Replacing a real number with a representable value introduces round-off error which is relatively small if the real number lies in the bounded range. However, there are always real numbers much larger and much smaller than any representable value, which cannot be approximated with small round-off error. It is common to treat such values as $\pm\infty$ and devise special arithmetic rules for them.

Binary computers keep *internal representations* of floating-point numbers as a string of binary digits, just like integers, but the bits are interpreted differently. A fixed number of bits give the sign and the base-two digits of the mantissa, while the rest give the sign and magnitude of the exponent, just as in scientific notation. Arithmetic algorithms for such strings of bits should be simple, so that hardware to compute sums, products, sign changes, comparisons and so on is as simple as possible. Unfortunately, there are many reasonable solutions to this design problem, so that different computers generally use different floating-point formats. Thus, different computers might have different sets of representable values, meaning they will produce different outputs even when running the same program on identical inputs.

However, it is possible to impose standards that control the maximum difference between the outputs of an algorithm on different machines. For example, the Standard C programming language requires each host computer to have a standard header file `float.h`, listing how many bits are devoted to the mantissa as well as the smallest and largest representable positive numbers in two common formats: float or single precision, and double precision.

Excerpt from `float.h`

```
#define     FLT_RADIX                    2
#define     FLT_ROUNDS                   1
#define     FLT_MANT_DIG                24
#define     FLT_MIN_EXP               -125
#define     FLT_MAX_EXP               +128
#define     FLT_MIN             1.17549435e-38
#define     FLT_MAX             3.40282347e+38
#define     FLT_EPSILON         1.19209290e-07
```

Radix is the base of the number system, here 2, for binary. *Rounds* indicates how the machine chooses representations for real numbers:

`-1`: no rounding is specified.

`0`: round toward 0. Choose the nearest representable value whose absolute value is no greater than the number.

`1`: round toward the nearest representable value. Ties are broken with a convention, for example, always choosing the representable value whose least significant mantissa digit is even. Such a rule makes the expected round-off error zero.

`2`: round toward $+\infty$, that is, round up. Choose the nearest representable value greater than or equal to the number.

`3`: round toward $-\infty$, that is, round down. Choose the nearest representable value less than or equal to the number.

Then there are three integers specifying how many digits there are in the mantissa, the minimum value, and the maximum value for the exponent.

Standard C also specifies the minimum and maximum *normalized* positive numbers representable by the computer, that is, those which have a nonzero first digit in the mantissa. Smaller positive numbers are representable by using the minimum exponent and small fractional parts, but these fall into a special class called *subnormal* numbers and have fewer digits of precision.

The *floating-point epsilon*, `FLT_EPSILON`, is a key measure of precision and truncation error. It is the difference between the floating-point representation of 1, which is exact, and that of the "next larger" representable floating-point number. On the example binary computer, this is $2^{-23} \approx 1.19209 \times 10^{-7}$, the value of the least significant bit in a 24-bit mantissa divided by the value of the most significant bit. This bounds the relative error of truncating a real number to the number of digits available to the computer. It shall be called ϵ_f in this text. A related quantity is the least positive number ϵ_0 such that the floating-point representation of $1 + \epsilon_0$ is different from that of 1. That is, the computer evaluates the comparison $1.0 + \epsilon_0 > 1.0$ as true, but considers $1.0 + \epsilon = 1.0$ for any positive ϵ less than ϵ_0. Any real number can be represented by the computer with a relative error less than ϵ_0. In all cases, $0 < \epsilon_0 \leq \epsilon_f$; on machines that round to the nearest representable value, $\epsilon_0 = \frac{1}{2}\epsilon_f$.

With this information, it is possible to compute the maximum error in a particular computation from particular inputs. Two implementations of an algorithm may be considered equivalent if, for sufficiently many and varied inputs, their outputs differ by no more than the sum of those maximum errors.

IEEE floating-point format

The Institute for Electrical and Electronics Engineers, or IEEE, sponsored a committee that in 1985 published a standard format for 32-bit binary floating-point computer arithmetic. The standard effectively defines a function $v : \mathbf{R} \to \mathbf{R}$ mapping any real number x to its nearest representable value $v(x)$. It does that by specifying how to store representable values as bit strings. Figure 1.2 shows how bits are allocated into three fields s, e, and f in that format.

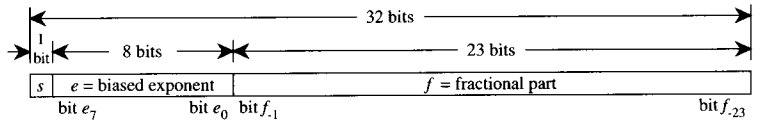


Figure 1.2: Schematic arrangement of bit fields in the single-precision (32-bit) IEEE binary floating-point format.

The s bit is 0 for positive numbers and 1 for negative numbers, that is, the sign of the represented value is that of $(-1)^s$. It is followed by an 8-bit exponent field in which an unsigned integer is stored, most-significant bit first, as $e = e_7 \cdots e_1 e_0$ (base 2). This e is called the *biased exponent*, and it takes the values 0,1,...,255. To obtain a signed value, a *bias* of 127 is subtracted to get the *unbiased exponent:*

$$E = e - 127 = e_7 \cdot 2^7 + \cdots + e_1 \cdot 2 + e_0 - 127. \tag{1.4}$$

Thus $-127 \leq E \leq +128$. However, the values $E = -127 \leftrightarrow e = 0$, as well as $E = +128 \leftrightarrow e = 255$, will be reserved to indicate special numbers like $\pm\infty$, so the valid range of unbiased exponents in this format is $-126 = E_{min} \leq E \leq E_{max} = +127$.

Let v be the real number represented by the bit strings s, e, f. Then v has the following interpretation:

- if $e = 255$ and $f \neq 0$, then v is *Not a Number (NaN)* regardless of the value of s. This value is considered different from $\pm\infty$ and can be used to signal invalid results.

- if $e = 255$ and $f = 0$, then $v = (-1)^s \infty$.

- if $0 < e < 255$, then $v = (-1)^s \, 2^{e-127} \, (1.f_{-1} \cdots f_{-23}$ (base 2)). This is also written $v = (-1)^s \, 2^E \, (1.f)$. Normalized mantissas must have nonzero first digit, so the fractional part of the number is supplied with a leading one: $1.f = 1 + f_{-1} \cdot 2^{-1} + \cdots + f_{-23} \cdot 2^{-23}$. This gives a precision of 24 bits while using only 23 actual bits. The extra bit of information is deduced from the exponent.

- if $e = 0$ and $f \neq 0$, then $v = (-1)^s \, 2^{-126} \, (0.f_{-1} \cdots f_{-23}$ (base 2)) is a *subnormal* or tiny number. Subnormal mantissas have leading digit zero: $0.f = f_{-1} \cdot 2^{-1} + \cdots + f_{-23} \cdot 2^{-23}$.

- if $e = 0$ and $f = 0$, then $v = (-1)^s \, 0$. For some operations, $+0$ and -0 are distinguished. In particular, $\sqrt{-0} = -0$ while $\sqrt{+0} = +0$. However, the comparison $-0 == +0$ evaluates as true on all machines conforming to this standard.

The floating-point format also specifies that certain arithmetic operations must be available, such as addition, division, remainder, and extraction of square roots.

Furthermore, it requires that the output of an operation be the representable value obtained by first performing the calculation as if all numbers were infinitely precise reals, and then rounding to the available precision. This is done by storing intermediate results of calculations in *extended* formats at higher precision than the final result. The IEEE requires extended formats to have at least a certain width, and also that either each representable number is encoded as a unique bit string, or else different bit strings representing the same value are not distinguished in any operation. These requirements ensure that any peculiarities of the loosely-specified extended formats remain invisible to the user, and guarantee that all computers conforming to the standard will produce identical results given identical inputs.

Conversion from decimal notation into the binary format are also treated by the standard, since many programs contain parameters entered by humans using scientific notation. Conversion is required to be *monotonic*, that is, if $x \geq y$ are two real numbers represented in scientific notation, then the associated representable value $v(x)$ must not be smaller than $v(y)$. All of the rounding procedures described in the previous section are monotonic.

There is also a *double precision* format in which the unbiased exponent contains 11 bits, the bias is 1023, there are 52 actual mantissa bits, and there is one sign bit. Conversion from single to double precision is exact, but converting from double to single precision involves rounding. There is likewise an *extended double precision* format. Computers with dedicated floating-point hardware usually perform all floating-point calculations in the extended double format, then round for output.

1.2.3 Propagation of error

In IEEE floating-point format with maximum representable value M =`FLT_MAX` and minimum normalized positive representable value m =`FLT_MIN`, the *normally representable set* is defined by $NR \stackrel{\text{def}}{=} [-M, -m] \cup [m, M]$. It consists of the nonzero real numbers which are approximated within relative error ϵ_f. Namely, suppose that $x_0 \in NR$ is a real number with nearest representable value $x = v(x_0)$. Then the *error of representation*, denoted $\Delta x = x - x_0$, satisfies

$$|\Delta x| \leq \epsilon_f \max\{|x|, |x_0|\} \leq \epsilon_f (1 + \epsilon_f)|x_0| \approx \epsilon_f |x_0|. \tag{1.5}$$

In a computer that rounds to the nearest representable value, $|\Delta x| \leq \frac{1}{2}\epsilon_f$, so as long as $0 < \epsilon_f \ll 1$ (it is typically 10^{-7}) we can use an honest inequality to estimate relative error:

$$\frac{|\Delta x|}{|x_0|} < \epsilon_f.$$

Henceforth it will be assumed that the computer rounds.

Note that if $x \in NR$, then $v(x)$ satisfies $v(v(x)) = v(x)$.

Conditioning

A computation $x \mapsto y$ is said to be *well-conditioned* if the relative error in y is comparable to the relative error in x. On a finite precision computer, the input x

can have a relative error as large as ϵ_f, so the conditioning of a single calculation "from exact inputs" is usually stated as the multiple of ϵ_f that we get for the relative error of y, given a relative error of ϵ_f in x.

When there are several inputs $(x_1, \ldots, x_n) \mapsto y$, it is assumed that they all have relative error ϵ_f, with the "worst" combination of signs.

An ill-conditioned calculation is one that can greatly magnify relative error.

Sums and differences

Suppose x, y, $x + y$, and $v(x) + v(y)$ all belong to NR. The IEEE procedure to compute $x + y$ is first to approximate x by $v(x)$ and y by $v(y)$, then to find $v(x) + v(y)$ in exact arithmetic, and finally to find the nearest representable value $v\left(v(x) + v(y)\right)$. The absolute error after this calculation is the difference between the exact value $x+y$ and the computed value $v\left(v(x) + v(y)\right)$. It can be estimated as follows, regardless of the rounding method used to obtain the representable value:

$$\begin{aligned} |x + y - v\left(v(x) + v(y)\right)| &\leq |x - v(x)| + |y - v(y)| \\ &\quad + |v(x) + v(y) - v\left(v(x) + v(y)\right)| \\ &< \epsilon_f \left(|x| + |y| + |x + y|\right). \end{aligned}$$

The three terms contributing to this error can differ greatly in magnitude. For example, if $x + y \approx 0$ but $x \approx -y \approx 1$, then $|x|$ and $|y|$ both are larger than $|x + y|$ and thus dominate the error estimate. In this case, it possible for the relative error in $x + y$ vastly to exceed the relative error in x or y. However, when x and y have the same sign, then there is no cancellation, so $|x| + |y| = |x + y|$ and the absolute error of the sum is less than $2(|x + y|)\epsilon_f$. The relative error of the sum is then no more than $2\epsilon_f$, the sum of the relative errors of x and y.

Products and quotients

If x, y, xy, and $v(x)v(y)$ all belong to NR, then an argument like that used to prove the product rule in calculus shows:

$$\begin{aligned} |xy - v(v(x)v(y))| &\leq |\left(x - v(x)\right)y| + |\left(y - v(y)\right)v(x)| \\ &\quad + |v(x)v(y) - v(v(x)v(y))| \\ &< 3\epsilon_f\, |xy|. \end{aligned}$$

The final step depends on Equation 1.5 and the assumption that $0 < \epsilon_f \ll 1$. Hence,

$$\left| \frac{xy - v(v(x)v(y))}{xy} \right| < 3\epsilon_f, \tag{1.6}$$

so the relative error in the product xy is no more than three times the maximum relative error ϵ_f of each of the factors. Products are therefore well-conditioned computations.

Quotients are products involving a reciprocal. The error of calculation for a reciprocal is

$$\left|\frac{1}{x} - v\left(\frac{1}{v(x)}\right)\right| \leq \left|\frac{1}{x} + v\left(\frac{1}{x}\right)\right| + \left|v\left(\frac{1}{x}\right) - v\left(\frac{1}{v(x)}\right)\right| < 2\epsilon_f \left|\frac{1}{x}\right|.$$

Here it is assumed that $x \neq 0$, $x \in NR$, and $1/v(x) \in NR$, as well as that 1 is exactly representable: $v(1) = 1$. The final step uses Equation 1.5 under the assumption that $0 < \epsilon_f \ll 1$. Hence the relative error of calculating the reciprocal is

$$\left|\frac{\frac{1}{x} - v\left(\frac{1}{v(x)}\right)}{\frac{1}{x}}\right| < 2\epsilon_f,$$

so each reciprocal in a product adds at most $2\epsilon_f$, rather than at most ϵ_f, to the relative error of the result. Combining the product and reciprocal formulas gives

$$\left|\frac{\frac{y}{x} - v\left(\frac{v(y)}{v(x)}\right)}{\frac{y}{x}}\right| < 4\epsilon_f. \tag{1.7}$$

Thus quotients, like products, are well-conditioned computations.

Functions

Suppose that $F = F(x)$ is a differentiable function. Any implementation of F actually computes $v(F(v(x)))$, since the computer converts any real-number input x into its representable value $v(x)$, and at best produces a representable output for the exact value $F(v(x))$, even if intermediate steps are exact. This error divides into two terms:

$$|F(x) - v(F(v(x)))| \leq |F(x) - F(v(x))| + |F(v(x)) - v(F(v(x)))|.$$

For the first term, we may put $\Delta x = x - v(x)$ and use the Mean Value Theorem of calculus to write

$$F(x) = F(v(x)) + F'(z)\,\Delta x, \tag{1.8}$$

where z is some number within $|\Delta x|$ of x. But $|\Delta x| < |x|\epsilon_f$, so if M is the maximum value of $|F'(z)|$ for z within $|x|\epsilon_f$ of x, we have $|F(x) - F(v(x))| \leq M|\Delta x| < M|x|\epsilon_f$. This also implies that $|F(v(x))| < |F(x)| + M|x|\epsilon_f$, so the second term has the bound $|F(v(x)) - v(F(v(x)))| < |F(v(x))|\epsilon_f < |F(x)|\epsilon_f + M|x|\epsilon_f^2$. These estimates combine to give $|F(x) - v(F(v(x)))| < (M'|x| + |F(x)|)\,\epsilon_f$, where $M' = (1 + \epsilon_f)M \approx M$. Since $\epsilon_f \ll 1$, dropping M' for M introduces negligible error, so we write (for $F(x) \neq 0$):

$$\left|\frac{F(x) - v(F(v(x)))}{F(x)}\right| < \left(1 + \frac{M|x|}{|F(x)|}\right)\epsilon_f. \tag{1.9}$$

Computing F is well-conditioned if $M|x| \approx |F(x)|$, but can be ill-conditioned if $M|x| \gg |F(x)|$.

If $x = (x_1, \ldots, x_n)$, and $F = F(x)$ is a differentiable function of several variables with gradient $\nabla F(x) = (\frac{\partial F}{\partial x_1}(x), \ldots \frac{\partial F}{\partial x_n}(x))$, then by the Mean Value Theorem we have

$$F(x) - F(v(x)) = \nabla F(z) \cdot \Delta x, \tag{1.10}$$

for some z on the line segment connecting x and $v(x) \stackrel{\text{def}}{=} (v(x_1), \ldots, v(x_n))$. Here Δx has components $\Delta x_k = x_k - v(x_k)$ satisfying $|\Delta x_k| \leq |x_k|\epsilon_f$. Writing M_k for the maximum value of $|\frac{\partial F}{\partial x_k}(z)|$ over all z between x and $v(x)$, we have

$$|F(x) - F(v(x))| \leq \epsilon_f \sum_{k=1}^{n} M_k |x_k|,$$

by the triangle inequality. The rest of the argument is similar to the one-variable case and gives the inequality

$$\left| \frac{F(x) - v(F(v(x)))}{F(x)} \right| < \left(1 + \frac{\sum_k M_k |x_k|}{|F(x)|} \right) \epsilon_f. \tag{1.11}$$

This inequality applies to products and quotients. For example, let $n = 2$ and put $F(x) = x_1 x_2$. Then $\nabla F(x) = (x_2, x_1)$, so $M_1 \approx |x_2|$ and $M_2 \approx |x_1|$ for z within $\epsilon_f \ll 1$ of x. The right-hand side of Inequality 1.11 simplifies to

$$\left(1 + \frac{|x_2 x_1| + |x_1 x_2|}{|x_2 x_1|} \right) \epsilon_f = 3\epsilon_f,$$

as in Inequality 1.6. The case of quotients is left as an exercise.

1.3 Exercises

1. Suppose a divides b and b divides a. Must $a = b$?

2. Write a computer program that finds the greatest common divisor of two integers a and b, assuming $b > a > 0$.

3. Prove that integer overflow or underflow occurs in w-bit twos complement integer arithmetic if and only if the carry into the sign bit is different from the carry out of the sign bit.

4. Express the integer 14 600 926 (base 10) in hexadecimal.

5. Prove that if $p \in \mathbf{Z}$ is a prime number, then $\sqrt{p}$ is not a rational number.

6. Write a computer program to read an integer in decimal notation and then print its binary digits and its hexadecimal digits. (Hint: most computers expect decimal number inputs and thus have built-in functions to read them.)

7. Convert the approximation $\pi \approx 3.1415926535897932$ (base 10) into the nearest 8-digit hexadecimal fraction.

8. Using 52 bits to represent the mantissa in IEEE binary floating-point format, how many decimal digits of accuracy are obtained?

9. What will $\sum_{k=1}^{10^8} 1.0$ equal on the example computer on p. 15, which uses IEEE 32-bit floating-point arithmetic?

10. Write a program to read 32-bit IEEE binary floating-point format and print the number in scientific notation. Have it treat NaN, $\pm\infty$, and ± 0 properly and have it signal when the number is subnormal.

11. Derive Inequality 1.7 from Inequality 1.11.

12. Determine and prove whether the following computations are well-conditioned or ill-conditioned:

    a. $(x, y) \mapsto \sqrt{x^2 + y^2}$, for $x \neq 0$ and $y \neq 0$

    b. $x \mapsto x \log x$, for $x > 0$

    c. $x \mapsto \lfloor x \rfloor$

1.4 Further Reading

- ANSI/IEEE. *Standard for Binary Floating-Point Arithmetic.* Document 754-1985, catalog number SH 10116-NYF. ISBN 1-55937-653-8.

- Donald Knuth. *Fundamental Algorithms*, volume 1. Addison-Wesley, Reading, Massachusetts, second edition, 1973. ISBN 0-201-03809-9.

- Behrooz Parhami. *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press, New York, 2000. ISBN 0-19-512583-5.

- Herbert Schildt. *The Annotated ANSI C Standard: ANSI/ISO 9899-1990.* Osborne Mcgraw Hill, Berkeley, California, 1993. ISBN 0-07-881952-0.

- Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, Boca Raton, Florida, 1995. ISBN 0-8493-8521-0.

Chapter 2

Space and Linearity

The ability to see the geometric properties of objects in space helps to visualize important properties of digital signals. It is only necessary to find the correspondence between the signal property and the geometric object. A starting point is the analytic geometry of the line, the plane, and space. However, most of the notions of geometry such as space, distance, angle, orientation, and motion can be defined much more generally,

Digital signals representing sounds and images are modeled by points in some of these generalized spaces, and many common transformations of such signals are easily described as geometric operations on those points. For example, points in space may be added together or multiplied by real numbers, which correspond respectively to mixing signals or amplifying them. The results are variously called linear combinations, superpositions, or linear transformations.

Linear transformations of linear spaces preserve linear combinations, in a sense that will be made precise. Even rather complicated transformations satisfy this linearity assumption, which in many cases reduces their analysis to linear algebra. This chapter will begin the rigorous analysis of signals and their transformations with an overview of the applicable linear algebra plus a few results from advanced calculus needed to understand infinite dimensional spaces.

2.1 Vector Spaces

Vectors started out as little arrows marking positions or displacements in space. They can be joined head-to-tail, moved about, rotated or stretched, and they define lengths, directions and angles. These properties, abstracted and listed under *Vector Space Axioms* in Section 2.1.2 below, are also possessed by functions, sequences, and matrices, all of which are very useful in digital signal processing.

Visualizing real numbers as a line boosts intuition. For example, $p > q$ means p lies to the right of q. Operations like $p \mapsto -p$ can be visualized as reflections to the other side of 0, and combinations like $(p, q) \mapsto \frac{1}{2}(p + q)$ replace two points with

their midpoint. The distance between p and q is $|p - q|$, and algebraic facts like $|p - q| \leq |p| + |q|$ are instantly suggested by comparing the direct path between p and q to the one that first passes by 0.

Points $\mathbf{p}$ in the plane are modeled by ordered pairs (p_1, p_2) of real numbers, after an origin $\mathbf{0} \stackrel{\text{def}}{=} (0, 0)$ and two perpendicular lines called *coordinate axes* are chosen. Coordinate p_1 gives the component of the point along one axis, p_2 the other.

An ordered pair $\mathbf{p} = (p_1, p_2)$ also models the head of an arrow or *vector* from $\mathbf{0}$. We can imagine sliding such an arrow around without rotation. Placing two vectors $\mathbf{p}$ and $\mathbf{q} = (q_1, q_2)$ head to tail produces a new vector $\mathbf{p} + \mathbf{q} \stackrel{\text{def}}{=} (p_1 + q_1, p_2 + q_2)$, giving a nice geometric model of addition of pairs of real numbers.

The distance between $\mathbf{p}$ and $\mathbf{q}$ is given by Pythagoras' theorem:

$$\|\mathbf{p} - \mathbf{q}\| \stackrel{\text{def}}{=} \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}.$$

Thus, a vector's length is $\|\mathbf{p}\| \stackrel{\text{def}}{=} \sqrt{p_1^2 + p_2^2}$. This length notation is intentionally similar to that for the line, since $\|\mathbf{p} - \mathbf{q}\| \leq \|\mathbf{p}\| + \|\mathbf{q}\|$ and like facts hold for points in the plane, too.

The *inner product* of two vectors in the plane is denoted by $\langle \mathbf{p}, \mathbf{q} \rangle \stackrel{\text{def}}{=} p_1 q_1 + p_2 q_2$. It is useful for measuring angles: If θ is the angle at the origin between two nonzero vectors $\mathbf{p}$ and $\mathbf{q}$, then the Law of Cosines can be used to show that

$$\cos\theta = \frac{\langle \mathbf{p}, \mathbf{q} \rangle}{\|\mathbf{p}\| \, \|\mathbf{q}\|}. \tag{2.1}$$

The arrows to $\mathbf{p}$ and $\mathbf{q}$ are perpendicular or *orthogonal* if and only if $\theta = 90°$, if and only if $\cos\theta = 0$, if and only if $\langle \mathbf{p}, \mathbf{q} \rangle = 0$.

Points and vectors in space are modeled by ordered triples $\mathbf{p} \stackrel{\text{def}}{=} (p_1, p_2, p_3)$. The length formula is $\|\mathbf{p}\| \stackrel{\text{def}}{=} \sqrt{p_1^2 + p_2^2 + p_3^2}$. Two nonzero points $\mathbf{p}$ and $\mathbf{q}$ and the origin $\mathbf{0} = (0, 0, 0)$ define a plane, and the angle at $\mathbf{0}$ between the vectors to $\mathbf{p}$ and $\mathbf{q}$ can be measured in this plane. If this angle is denoted by θ, it may be computed from Equation 2.1, where now $\langle \mathbf{p}, \mathbf{q} \rangle \stackrel{\text{def}}{=} p_1 q_1 + p_2 q_2 + p_3 q_3$.

Notice that in all of these examples, $\|\mathbf{p}\| = \sqrt{\langle \mathbf{p}, \mathbf{p} \rangle}$.

2.1.1 Euclidean space

While not readily visualized, space can be generalized to a set of ordered lists of more than three real numbers. Fix a positive integer *dimension* N and consider $\mathbf{R}^N$, the set of ordered lists of N real numbers. Elements $\mathbf{p} = (p_1, \ldots, p_N)$ and $\mathbf{q} = (q_1, \ldots, q_N)$ can be combined componentwise, and they have lengths and angles analogous to points and vectors in more easily visualized systems. Namely,

- $\mathbf{p} + \mathbf{q} \stackrel{\text{def}}{=} (p_1 + q_1, \ldots, p_N + q_N)$;
- $c\mathbf{p} \stackrel{\text{def}}{=} (cp_1, \ldots, cp_N)$, for any real number c;

Vertex	Coordinates	Edge to	Vertex	Coordinates	Edge to
$\mathbf{P}_0$	$(0,0,0)$	$\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_4$	$\mathbf{P}_3$	$(0,1,1)$	$\mathbf{P}_7$
$\mathbf{P}_1$	$(0,0,1)$	$\mathbf{P}_3$, $\mathbf{P}_5$	$\mathbf{P}_5$	$(1,0,1)$	$\mathbf{P}_7$
$\mathbf{P}_2$	$(0,1,0)$	$\mathbf{P}_3$, $\mathbf{P}_6$	$\mathbf{P}_6$	$(1,1,0)$	$\mathbf{P}_7$
$\mathbf{P}_4$	$(1,0,0)$	$\mathbf{P}_5$, $\mathbf{P}_6$	$\mathbf{P}_7$	$(1,1,1)$	

Table 2.1: Vertices and edges of the three-dimensional unit cube.

Vertex	Coordinates	Edge to	Vertex	Coordinates	Edge to
$\mathbf{Q}_0$	$(0,0,0,0)$	$\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{Q}_4$, $\mathbf{Q}_8$	$\mathbf{Q}_9$	$(1,0,0,1)$	$\mathbf{Q}_{11}$, $\mathbf{Q}_{13}$
$\mathbf{Q}_1$	$(0,0,0,1)$	$\mathbf{Q}_3$, $\mathbf{Q}_5$, $\mathbf{Q}_9$	$\mathbf{Q}_{10}$	$(1,0,1,0)$	$\mathbf{Q}_{11}$, $\mathbf{Q}_{14}$
$\mathbf{Q}_2$	$(0,0,1,0)$	$\mathbf{Q}_3$, $\mathbf{Q}_6$, $\mathbf{Q}_{10}$	$\mathbf{Q}_{12}$	$(1,1,0,0)$	$\mathbf{Q}_{13}$, $\mathbf{Q}_{14}$
$\mathbf{Q}_4$	$(0,1,0,0)$	$\mathbf{Q}_5$, $\mathbf{Q}_6$, $\mathbf{Q}_{12}$	$\mathbf{Q}_7$	$(0,1,1,1)$	$\mathbf{Q}_{15}$
$\mathbf{Q}_8$	$(1,0,0,0)$	$\mathbf{Q}_9$, $\mathbf{Q}_{10}$, $\mathbf{Q}_{12}$	$\mathbf{Q}_{11}$	$(1,0,1,1)$	$\mathbf{Q}_{15}$
$\mathbf{Q}_3$	$(0,0,1,1)$	$\mathbf{Q}_7$, $\mathbf{Q}_{11}$	$\mathbf{Q}_{13}$	$(1,1,0,1)$	$\mathbf{Q}_{15}$
$\mathbf{Q}_5$	$(0,1,0,1)$	$\mathbf{Q}_7$, $\mathbf{Q}_{13}$	$\mathbf{Q}_{14}$	$(1,1,1,0)$	$\mathbf{Q}_{15}$
$\mathbf{Q}_6$	$(0,1,1,0)$	$\mathbf{Q}_7$, $\mathbf{Q}_{14}$	$\mathbf{Q}_{15}$	$(1,1,1,1)$	

Table 2.2: Vertices and edges of the four-dimensional unit cube.

- $\|\mathbf{p}\| \stackrel{\text{def}}{=} \sqrt{p_1^2 + \cdots + p_N^2}$;
- $\langle \mathbf{p}, \mathbf{q} \rangle \stackrel{\text{def}}{=} p_1 q_1 + \cdots p_N q_N$.

The set $\mathbf{R}^N$ equipped with these operations will be called *N-dimensional Euclidean space*, or *Euclidean N-space*. Note that $\|\mathbf{p}\| = \sqrt{\langle \mathbf{p}, \mathbf{p} \rangle}$, and $\langle \mathbf{p}, \mathbf{q} \rangle = \frac{1}{4}\left(\|\mathbf{p}+\mathbf{q}\|^2 - \|\mathbf{p}-\mathbf{q}\|^2\right)$.

For example, the eight corner vertices of a unit cube in space are listed in Table 2.1, numbered by reading their coordinates in binary. There are 12 edges, one joining each pair of vertices that differ in exactly one coordinate: $\overline{\mathbf{P}_0\mathbf{P}_1}$, $\overline{\mathbf{P}_0\mathbf{P}_2}$, $\overline{\mathbf{P}_0\mathbf{P}_4}$; $\overline{\mathbf{P}_1\mathbf{P}_3}$, $\overline{\mathbf{P}_1\mathbf{P}_5}$; $\overline{\mathbf{P}_2\mathbf{P}_3}$, $\overline{\mathbf{P}_2\mathbf{P}_6}$; $\overline{\mathbf{P}_4\mathbf{P}_5}$, $\overline{\mathbf{P}_4\mathbf{P}_6}$; $\overline{\mathbf{P}_3\mathbf{P}_7}$; $\overline{\mathbf{P}_5\mathbf{P}_7}$; $\overline{\mathbf{P}_6\mathbf{P}_7}$.

Similarly, the 16 corner vertices and 32 edges of a unit cube in Euclidean 4-space are listed in Table 2.2. To count edges, we imagine making the 4-cube by sweeping a 3-cube along a fourth axis. That creates eight new edges, joining corresponding vertices of the front and back 3-cubes. The front and back 3-cubes have 12 edges each, for a total of 32.

Generalizing $\mathbf{R}^N$ is the set $\mathbf{C}^N$ of ordered lists of N complex numbers. In this case, the *Hermitean inner product* is used:

$$\langle \mathbf{p}, \mathbf{q} \rangle \stackrel{\text{def}}{=} \bar{p}_1 q_1 + \cdots \bar{p}_N q_N. \tag{2.2}$$

The complex conjugate on the first vector insures that $\|\mathbf{p}\| \stackrel{\text{def}}{=} \sqrt{\langle \mathbf{p}, \mathbf{p} \rangle}$ is a non-negative real number. $\mathbf{C}^N$ with the Hermitean inner product will be called *complex N-dimensional Euclidean space*, though it may also be regarded as $2N$-dimensional Euclidean space simply by treating the real and imaginary parts of each coordinate as separate coordinates in themselves. Note that the real and imaginary parts

of the complex-valued inner product are $\Re\langle \mathbf{p}, \mathbf{q}\rangle = \frac{1}{4}\left(\|\mathbf{p}+\mathbf{q}\|^2 - \|\mathbf{p}-\mathbf{q}\|^2\right)$ and $\Im\langle \mathbf{p}, \mathbf{q}\rangle = \frac{1}{4}\left(\|i\mathbf{p}+\mathbf{q}\|^2 - \|i\mathbf{p}-\mathbf{q}\|^2\right)$, so

$$\langle \mathbf{p}, \mathbf{q}\rangle = \frac{1}{4}\left(\|\mathbf{p}+\mathbf{q}\|^2 - \|\mathbf{p}-\mathbf{q}\|^2\right) + \frac{i}{4}\left(\|i\mathbf{p}+\mathbf{q}\|^2 - \|i\mathbf{p}-\mathbf{q}\|^2\right).$$

This is sometimes called the *polarization identity*. Note that the imaginary part vanishes if the coordinates of $\mathbf{p}, \mathbf{q}$ are all purely real or all purely imaginary.

Vectors in $\mathbf{R}^N$ or $\mathbf{C}^N$ are sometimes displayed as columns of numbers, or *column vectors*, such as

$$\mathbf{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_N \end{pmatrix}; \quad \mathbf{q} = \begin{pmatrix} q_1 \\ \vdots \\ q_N \end{pmatrix}.$$

The *adjoint* $\mathbf{p}^* \stackrel{\text{def}}{=} (\bar{p}_1 \ \ldots \ \bar{p}_N)$ of a column vector $\mathbf{p}$ is the *row vector* filled with the complex conjugates of the coordinates of $\mathbf{p}$. The Hermitean inner product can be written in terms of matrix multiplication, defined in Equation 2.41 further on, and this adjoint:

$$\langle \mathbf{p}, \mathbf{q}\rangle = \mathbf{p}^*\mathbf{q}. \tag{2.3}$$

This is also a special case of Equation 2.47.

From this point on, we will use $\mathbf{E}^N$ to denote both real and complex N-dimensional Euclidean space, with the understanding that $\mathbf{E}^N = \mathbf{R}^N$ if the coordinates are real numbers and $\mathbf{E}^N = \mathbf{C}^N$ if they are complex numbers.

2.1.2 Abstract vector spaces

A *vector space* is the further abstraction and generalization of Euclidean N-space. It consists of a set $\mathbf{X}$ of *vectors* that can be added together and multiplied by *scalars* (real or complex numbers) to form *linear combinations*, which are again elements of $\mathbf{X}$. For example, if $\mathbf{x}, \mathbf{y}$ are vectors in $\mathbf{X}$ and a and b are two fixed scalars, then the linear combination $a\mathbf{x} + b\mathbf{y}$ also belongs to $\mathbf{X}$. It follows by induction that a linear combination of finitely many vectors from $\mathbf{X}$ must also belong to $\mathbf{X}$. The following rules, as in Euclidean space, apply to all vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$ and scalars a, b:

Vector Space Axioms

Associativity: $\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$.

Commutativity: $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$.

Identity: There is a unique element $\mathbf{0} \in \mathbf{X}$ called the *identity*, satisfying $\mathbf{x} + \mathbf{0} = \mathbf{0} + \mathbf{x} = \mathbf{x}$ for all $\mathbf{x} \in \mathbf{X}$.

Inverses: Each $\mathbf{x} \in \mathbf{X}$ has a unique *inverse* $-\mathbf{x} \stackrel{\text{def}}{=} (-1)\mathbf{x} \in \mathbf{X}$, satisfying $\mathbf{x} + (-\mathbf{x}) = -\mathbf{x} + \mathbf{x} = \mathbf{0}$.

Distributivity: $a(\mathbf{x} + \mathbf{y}) = a\mathbf{x} + a\mathbf{y}$.

Scalar multiplication associativity: $(ab)\mathbf{x} = a(b\mathbf{x})$.

A *subspace* $\mathbf{Y}$ of a vector space $\mathbf{X}$ is any nonempty subset $\mathbf{Y} \subset \mathbf{X}$ which is closed under addition and scalar multiplication: $\mathbf{u} + c\mathbf{v} \in \mathbf{Y}$ for any $\mathbf{u}, \mathbf{v} \in \mathbf{Y}$ and any scalar c. For example, $\mathbf{X}$ is a subspace of itself. The zero subspace $\{\mathbf{0}\}$ is another example. Every subspace $\mathbf{Y} \subset \mathbf{X}$ contains $\mathbf{0}$, since we may take $c = -1$ and $\mathbf{u} = \mathbf{v}$. Likewise, taking $\mathbf{u} = \mathbf{0}$ and $c = -1$ proves that $-\mathbf{v} \in \mathbf{Y}$ whenever $\mathbf{v} \in \mathbf{Y}$.

The *sum of subspaces* $\mathbf{Y}, \mathbf{Z}$ of a vector space $\mathbf{X}$ is the set of sums:

$$\mathbf{Y} + \mathbf{Z} \stackrel{\text{def}}{=} \{\mathbf{y} + \mathbf{z} : \quad \mathbf{y} \in \mathbf{Y}, \mathbf{z} \in \mathbf{Z}\}. \tag{2.4}$$

We can have sums of many subspaces, such as $\mathbf{Y}_1 + \mathbf{Y}_2 + \cdots + \mathbf{Y}_m$.

If $\mathbf{X}$ is a vector space and $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ is any finite set of vectors from $\mathbf{X}$, then the *linear span* $\operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ is a subspace of $\mathbf{X}$, consisting of all linear combinations of $\mathbf{v}_1, \ldots, \mathbf{v}_n$:

$$\operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \stackrel{\text{def}}{=} \left\{ \sum_{i=1}^{n} a_i \mathbf{v}_i : \ a_1, \ldots, a_n \text{ are scalars.} \right\} \tag{2.5}$$

We observe that

$$\operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_n\} = \operatorname{span}\{\mathbf{v}_1\} + \cdots + \operatorname{span}\{\mathbf{v}_n\}. \tag{2.6}$$

A set of vectors $\{\mathbf{v}_1, \cdots, \mathbf{v}_n\}$ in a vector space $\mathbf{X}$ is called *linearly dependent* if there are scalars $a_1, \ldots, a_n$, not all zero, for which $a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n = \mathbf{0}$. This nontrivial linear combination representing zero is called a *dependency relation.* Otherwise, if no such relation exists, then we say that the set $\{\mathbf{v}_1, \cdots, \mathbf{v}_n\} \subset \mathbf{X}$ is *linearly independent.* Given linearly independent vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ and told that $a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n = \mathbf{0}$ for scalars $a_1, \ldots, a_n$, we may conclude that $a_1 = \cdots = a_n = 0$.

The scalars $c_1, \ldots, c_n$ in the linear combination $\mathbf{x} = c_1\mathbf{v}_1 + \cdots + c_n\mathbf{v}_n \in \mathbf{X}$ are called the *expansion coefficients* of $\mathbf{x}$ in terms of the set $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$. Linear independence guarantees that expansions are unique: if $\mathbf{x} = c_1\mathbf{v}_1 + \cdots + c_n\mathbf{v}_n = c'_1\mathbf{v}_1 + \cdots + c'_n\mathbf{v}_n$, we may conclude that $c_k = c'_k$ for all $k = 1, \ldots, n$.

The N *standard basis vectors* of $\mathbf{E}^N$ are linearly independent:

$$\mathbf{e}_1 \stackrel{\text{def}}{=} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \mathbf{e}_2 \stackrel{\text{def}}{=} \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \quad \ldots, \quad \mathbf{e}_N \stackrel{\text{def}}{=} \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}. \tag{2.7}$$

Here $\mathbf{e}_n$ has a single 1 in row n, and zeroes in all other rows. The vector $a_1\mathbf{e}_1 + \cdots + a_N\mathbf{e}_N$ has a_n at position n, and linear independence is an immediate conclusion. A more complicated example is:

$$\mathbf{f}_1 \stackrel{\text{def}}{=} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \mathbf{f}_2 \stackrel{\text{def}}{=} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \quad \ldots, \quad \mathbf{f}_N \stackrel{\text{def}}{=} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}. \tag{2.8}$$

Thus $\mathbf{f}_1 = \mathbf{e}_1$, $\mathbf{f}_2 = \mathbf{e}_1 + \mathbf{e}_2$, and $\mathbf{f}_N = \mathbf{e}_1 + \mathbf{e}_2 + \cdots + \mathbf{e}_N$. If $a_1\mathbf{f}_1 + \cdots + a_N\mathbf{f}_N = \mathbf{0}$, we immediately conclude that $a_N = 0$, so in fact $a_1\mathbf{f}_1 + \cdots + a_{N-1}\mathbf{f}_{N-1} = \mathbf{0}$. But that inductive step reduces the problem to the case of $N-1$ vectors, and the case $N = 1$ is evidently true, so linear independence follows.

The *dimension* of a vector space $\mathbf{X}$ is the largest number of vectors in any linearly independent subset of $\mathbf{X}$. If there are linearly independent sets of N vectors for any N, then the space is called *infinite dimensional*.

Finite dimensions

An N-dimensional vector space $\mathbf{X}$ must contain a set $\{\mathbf{b}_1, \ldots, \mathbf{b}_N\}$ of N linearly independent vectors. Any such set is called a *basis* for $\mathbf{X}$.

Theorem 2.1 *Suppose $\{\mathbf{b}_1, \ldots, \mathbf{b}_N\}$ is a basis for the N-dimensional vector space $\mathbf{X}$. Then each vector $\mathbf{x} \in \mathbf{X}$ can be expressed as a linear combination of vectors $\{\mathbf{b}_n\}$ in one and only one way. That is, for each $\mathbf{x} \in \mathbf{X}$ there is a unique set of N scalars $c_1, \ldots, c_N$, depending on $\mathbf{x}$, giving the expansion*

$$\mathbf{x} = \sum_{n=1}^{N} c_n \mathbf{b}_n.$$

Proof: For existence, suppose that there is a vector $\mathbf{x} \in \mathbf{X}$ not expressible as such a linear combination. We can be sure that $\mathbf{x} \neq \mathbf{0} = \sum_n 0\mathbf{b}_n$, and so $\{\mathbf{x}\} \cup \{\mathbf{b}_1, \ldots, \mathbf{b}_N\} \subset \mathbf{X}$ must be a linearly independent set of $N+1$ vectors. This contradicts the assumption that $\mathbf{X}$ is N-dimensional.

The uniqueness of the expansion follows from the linear independence of the set $\{\mathbf{b}_n\}$: if $\mathbf{x} = \sum_n c_n\mathbf{b}_n = \sum_n c'_n\mathbf{b}_n$, then $\sum_n (c'_n - c_n)\mathbf{b}_n = \mathbf{0}$, so $c'_n = c_n$ for all $n = 1, \ldots, N$. □

Not surprisingly, N-dimensional Euclidean space has dimension N, but this fact requires proof. The N standard basis vectors are linearly independent, so the dimension is at least N, but we must show that no larger linearly independent set exists. Since Euclidean N-space is span $\{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$, it suffices to show:

Theorem 2.2 *Suppose that $\mathbf{v}_1, \ldots, \mathbf{v}_N$ are N vectors in some vector space. Then any set of $N+1$ or more vectors in* span $\{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$ *must be linearly dependent.*

Proof: We first note that if all sets of M vectors are linearly dependent, then any set of more than M must also be linearly dependent: the dependency relation for the first M can be padded with zero coefficients for the remaining vectors.

We proceed by induction on N. For $N = 1$, given $\mathbf{x}, \mathbf{y} \in$ span $\{\mathbf{v}\}$, write $\mathbf{x} = a\mathbf{v}$ and $\mathbf{y} = b\mathbf{v}$. If a, b are not both zero, we can put $a' = b$ and $b' = -a$ to get $a'\mathbf{x} + b'\mathbf{y} = (ab - ba)\mathbf{v} = \mathbf{0}$ with a', b' not both zero. Otherwise if $a = b = 0$, then $a' = b' = 1$ gives a dependency relation. Thus, every set of two (or more) vectors in the span of a single vector is linearly dependent.

For $N \geq 2$, we make the inductive hypothesis that every set of N or more vectors in the span of $N-1$ vectors is linearly dependent. Now suppose that the

$N+1$ vectors $\mathbf{W} \stackrel{\text{def}}{=} \{\mathbf{w}_1, \ldots, \mathbf{w}_{N+1}\}$ belong to span $\{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$. Then we may write

$$\begin{aligned} \mathbf{w}_1 &= a(1,1)\mathbf{v}_1 + \cdots + a(1,N)\mathbf{v}_N, \\ &\vdots \\ \mathbf{w}_N &= a(N,1)\mathbf{v}_1 + \cdots + a(N,N)\mathbf{v}_N, \\ \mathbf{w}_{N+1} &= a(N+1,1)\mathbf{v}_1 + \cdots + a(N+1,N)\mathbf{v}_N, \end{aligned}$$

where each $a(i,j)$ is a scalar. If the last column of scalars is all zero, namely $a(i,N) = 0$ for $i = 1, \ldots, N+1$, then in fact $\mathbf{W} \subset \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{N-1}\}$ and must be linearly dependent by the inductive hypothesis. Otherwise, we may suppose, by renumbering if necessary, that $a(N+1,N) \neq 0$, and eliminate the $\mathbf{v}_N$ terms as follows:

$$\mathbf{W}' \stackrel{\text{def}}{=} \left\{ \mathbf{w}'_i = \mathbf{w}_i - \frac{a(i,N)}{a(N+1,N)}\mathbf{w}_{N+1} : \; i = 1, \ldots, N \right\}. \tag{2.9}$$

This set $\mathbf{W}' \subset \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{N-1}\}$ is linearly dependent by the inductive hypothesis. But any dependency relation $c_1\mathbf{w}'_1 + \cdots + c_N\mathbf{w}'_N = \mathbf{0}$ for $\mathbf{W}'$ gives a dependency relation $c_1\mathbf{w}_1 + \cdots + c_N\mathbf{w}_N + b\mathbf{w}_{N+1} = \mathbf{0}$ for $\mathbf{W}$, where $b = -\sum_i c_i a(i,N)/a(N+1,N)$. □

Any subspace of an N-dimensional vector space $\mathbf{X}$ is finite dimensional with dimension at most N. For example, the set of vectors in $\mathbf{E}^N$ with zero first coordinate is an $(N-1)$-dimensional subspace of $\mathbf{E}^N$. Vectors $\mathbf{e}_2, \ldots, \mathbf{e}_N$ are a basis for that subspace.

Norms

A *norm* in a vector space $\mathbf{X}$ is a nonnegative real-valued function $\mathbf{x} \mapsto \|\mathbf{x}\|$ satisfying the following:

Norm Axioms

Sublinearity: $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\|+\|\mathbf{y}\|$ and $\|a\mathbf{x}\| = |a|\,\|\mathbf{x}\|$ for any $\mathbf{x}, \mathbf{y} \in \mathbf{X}$ and any scalar a.

Nondegeneracy: $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$, the additive identity, or zero vector; otherwise, $\|\mathbf{x}\| > 0$.

Any N-dimensional vector space $\mathbf{X}$ can be equipped with a norm. Start by fixing a basis $\mathbf{b}_1, \ldots, \mathbf{b}_N$, so that any $\mathbf{x} \in \mathbf{X}$ is expressible in one and only one way as $\mathbf{x} = \sum_{n=1}^N x(n)\mathbf{b}_n$. Then, let

$$\|\mathbf{x}\|_1 \stackrel{\text{def}}{=} |x(1)| + \cdots + |x(N)|, \tag{2.10}$$

where $\mathbf{x} = \sum_n x(n)\mathbf{b}_N$ is the unique expansion of $\mathbf{x}$. It is a straightforward exercise to show that this satisfies the norm axioms. Alternatively, we can mimic the formula from Euclidean N-space:

$$\|\mathbf{x}\|_2 \stackrel{\text{def}}{=} \sqrt{|x(1)|^2 + \cdots + |x(N)|^2}. \tag{2.11}$$

In fact, for any $p \geq 1$, there is a so-called p-norm:

$$\|\mathbf{x}\|_p \stackrel{\text{def}}{=} (|x(1)|^p + \cdots + |x(N)|^p)^{\frac{1}{p}}. \tag{2.12}$$

The formula extends to the limit $p \to \infty$:

$$\|\mathbf{x}\|_\infty \stackrel{\text{def}}{=} \max\{|x(1)|, \ldots, |x(N)|\}. \tag{2.13}$$

But the choice of norm does not matter too much in a finite dimensional vector space. For example, the norms $\|\cdot\|_1, \|\cdot\|_2, \|\cdot\|_\infty$ defined on $\mathbf{E}^N$ in terms of the standard basis vectors $\{\mathbf{e}_n\}$ satisfy the following system of inequalities:

$$\|\mathbf{x}\|_1 \leq \sqrt{N}\,\|\mathbf{x}\|_2 \qquad \|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1 \tag{2.14}$$

$$\|\mathbf{x}\|_1 \leq N\,\|\mathbf{x}\|_\infty \qquad \|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_1 \tag{2.15}$$

$$\|\mathbf{x}\|_2 \leq \sqrt{N}\|\mathbf{x}\|_\infty \qquad \|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \tag{2.16}$$

In fact, all norms on an N-dimensional vector space satisfy similar inequalities:

Theorem 2.3 *Any two norms $\|\cdot\|_p$ and $\|\cdot\|_q$ for an N-dimensional vector space $\mathbf{X}$ are comparable. That is, there are positive numbers A, B, C, D such that for all $\mathbf{x} \in \mathbf{X}$,*

$$A\|\mathbf{x}\|_p \leq \|\mathbf{x}\|_q \leq B\|\mathbf{x}\|_p, \quad \text{and} \quad C\|\mathbf{x}\|_q \leq \|\mathbf{x}\|_p \leq D\|\mathbf{x}\|_q.$$

Proof: Fixing a basis $\{\mathbf{b}_n : n = 1, \ldots, N\} \subset \mathbf{X}$, we will show that the norm $\|\cdot\|_\infty$ defined by Equation 2.13 is comparable to all others. Let $\|\cdot\|$ be any other norm on $\mathbf{X}$. Then by its sublinearity,

$$\|\mathbf{x}\| \leq \sum_{n=1}^{N} \|x(n)\mathbf{b}_n\| = \sum_{n=1}^{N} |x(n)|\,\|\mathbf{b}_n\| \leq \left(\max_{1\leq n\leq N} |x(n)|\right)\left(\sum_{n=1}^{N} \|\mathbf{b}_n\|\right). \tag{2.17}$$

So $\|\mathbf{x}\| \leq B\|\mathbf{x}\|_\infty$ for $B = \|\mathbf{b}_1\| + \cdots + \|\mathbf{b}_N\| > 0$. We know that $B > 0$ because $\|\cdot\|$ is nondegenerate and basis vectors are nonzero.

For the other inequality, suppose toward contradiction that there is no $A > 0$ such that $A\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|$ for every $\mathbf{x}$. Then there must be an infinite sequence of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots\} \subset \mathbf{X}$ with the property that $\|\mathbf{x}_k\|_\infty = 1$ for all $k = 1, 2, \ldots$, but $\|\mathbf{x}_k\| \to 0$ as $k \to \infty$. We may suppose without loss of generality that the expansion coefficients $(x_k(1), \ldots, x_k(N))$ of $\mathbf{x}_k = \sum_{n=1}^{N} x_k(n)\mathbf{b}_n$, $k = 1, 2, \ldots$ are real numbers and define a point in $\mathbf{R}^N$, since N complex scalars are the same as $2N$ real scalars. These real coordinates are at most 1 in absolute value, so $\{\mathbf{x}_k : k = 1, 2, \ldots\}$ gives an infinite sequence of points confined to the unit hypercube $[-1, 1]^N \subset \mathbf{R}^N$. Cutting this hypercube in half along each axis gives 2^N subcubes, at least one of which must contain infinitely many[1] of the points $\{\mathbf{x}_k\}$. For notational convenience, let us suppose without loss of generality that it is the subcube $[0, 1]^N$ of nonnegative

[1] How can we be sure that such a packed subcube exists? Why does this proof fail in infinite dimensional space?

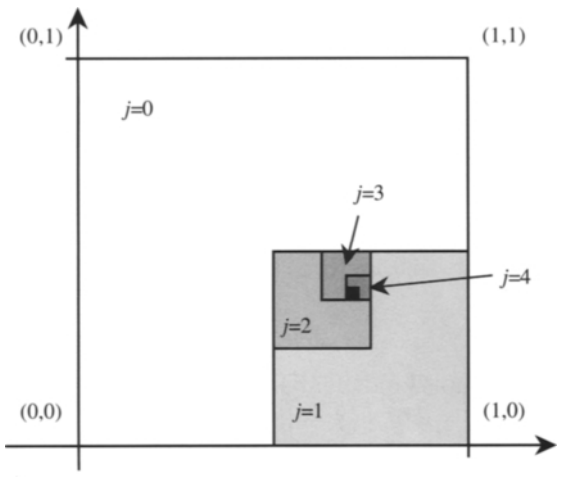


Figure 2.1: Nested subcubes at stages $j = 0, 1, 2, 3, 4$, to illustrate the two-dimensional case of Theorem 2.3.

coordinates. Call this the *stage* 0 *subcube*, and let $\mathbf{y}_0 \in \mathbf{R}^N$ be one of its points, say $\mathbf{x}_{k(0)}$.

We next define the *stage* j *subcube* for $j > 0$ by cutting the stage $j-1$ subcube in half along each axis and picking one of its 2^N subcubes, making sure that the chosen one contains infinitely many of the points $\{\mathbf{x}_k\}$. An example with $N = 2$ and $j = 0, 1, 2, 3, 4$ is depicted in Figure 2.1. We define $\mathbf{y}_j$ to be one of those points, say $\mathbf{x}_{k(j)}$. We can make the choice such that $k(0) < k(1) < k(2) < \cdots$. Now notice that the n^{th} coordinates of the points $\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \ldots$ form a Cauchy sequence, since the first j binary digits of $y_j(n), y_{j+1}(n), \ldots$ are all the same. By the completeness of the reals, Theorem 1.9, there is a unique real number $y(n) = \lim_{j\to\infty} y_j(n)$ for $n = 1, 2 \ldots, N$. Putting $\mathbf{x} \stackrel{\text{def}}{=} \sum_{n=1}^{N} y(n)\mathbf{b}_n \in \mathbf{X}$, we see that $\|\mathbf{x}\|_\infty = 1$ since $\|\mathbf{x}_k\|_\infty = 1$ for all k, including $k = k(j)$, and

$$\left| \|\mathbf{x}_{k(j)}\|_\infty - \|\mathbf{x}\|_\infty \right| \le \|\mathbf{x}_{k(j)} - \mathbf{x}\|_\infty = \max_{1\le n\le N} |y_j(n) - y(n)| \to 0,$$

as $j \to \infty$. Therefore $\mathbf{x} \neq \mathbf{0}$. On the other hand, we have the contradiction $\|\mathbf{x}\| = 0$, since $\|\mathbf{x}_{k(j)}\| \to 0$ as $j \to \infty$ by assumption, and

$$\left| \|\mathbf{x}_{k(j)}\| - \|\mathbf{x}\| \right| \le \|\mathbf{x}_{k(j)} - \mathbf{x}\| \le B\|\mathbf{x}_{k(j)} - \mathbf{x}\|_\infty = B \max_{1\le n\le N} |y_j(n) - y(n)| \to 0,$$

as $j \to \infty$, using Inequality 2.17. Hence $\|\cdot\|$ cannot be arbitrarily small relative to $\|\cdot\|_\infty$: there must be some $A > 0$ such that $A\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|$ for every $\mathbf{x}$.

To finish the proof, given any two norms $\|\cdot\|_p$ and $\|\cdot\|_q$ we find $A_p, B_p, A_q, B_q > 0$ such that

$$A_p\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_p \le B_p\|\mathbf{x}\|_\infty, \quad \text{and} \quad A_q\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_q \le B_q\|\mathbf{x}\|_\infty.$$

But then $A = A_q/B_p$, $B = B_q/A_p$, $C = A_p/B_q$, and $D = B_p/A_q$ satisfy the theorem. □

If $\mathbf{X}$ is an N-dimensional vector space and $\mathbf{B} = \{\mathbf{b}_n : n = 1, \ldots, N\} \subset \mathbf{X}$ is a basis, then every $\mathbf{x} = \sum_n c(n)\mathbf{b}_n \in \mathbf{X}$ corresponds to one and only one $\mathbf{c} = (c(1), \ldots, c(N)) \in \mathbf{E}^N$:

$$\mathbf{X} \overset{\mathbf{B}}{\longleftrightarrow} \mathbf{E}^N. \tag{2.18}$$

Having the freedom to choose $\mathbf{B}$ can sometimes simplify calculations in $\mathbf{X}$, which in practice must be done by mapping vectors $\mathbf{x}$ to points in $\mathbf{E}^N$.

Infinite dimensions

We will only consider a simple kind of infinite dimensional vector space $\mathbf{X}$, one which has a norm and a basis $\mathbf{B} = \{\mathbf{b}_n : n = 1, 2, \ldots\} \subset \mathbf{X}$ satisfying the following:

Schauder Basis Axioms

Linear independence: Any finite subset of the vectors in $\mathbf{B}$ is linearly independent.

Completeness: Each $\mathbf{x} \in \mathbf{X}$ has a $\mathbf{B}$-expansion $\mathbf{x} = \sum_n c(n)\mathbf{b}_n$.

Unique representation: If $\mathbf{x} = \sum_n c(n)\mathbf{b}_n$ and $\mathbf{x} = \sum_n c'(n)\mathbf{b}_n$, then $c(n) = c'(n)$ for all n.

Completeness in an infinite dimensional vector space $\mathbf{X}$ is interpreted to mean that for every $\mathbf{x} \in \mathbf{X}$, there is a sequence $\{c(n) : n = 1, 2, \ldots\}$ of expansion coefficients such that

$$\lim_{N \to \infty} \left\| \mathbf{x} - \sum_{n=1}^{N} c(n)\mathbf{b}_n \right\| = 0.$$

Alternatively, a set of vectors $\mathbf{B} \subset \mathbf{X}$ is called *dense* in $\mathbf{X}$ if, for every fixed vector $\mathbf{x} \in \mathbf{X}$ and $\epsilon > 0$, there is a finite linear combination $\mathbf{x}_\epsilon$ of vectors in $\mathbf{B}$ satisfying $\|\mathbf{x} - \mathbf{x}_\epsilon\| < \epsilon$. It is clear that a Schauder basis $\mathbf{B}$ is dense, but a set which is not a basis may also be dense.

We may extend the definition of linear span to arbitrary subsets of arbitrary vector spaces $\mathbf{X}$: for any $\mathbf{B} \subset \mathbf{X}$,

$$\operatorname{span} \mathbf{B} \overset{\text{def}}{=} \left\{ \sum_{n=1}^{N} a(n)\mathbf{b}_n : N \geq 0; \mathbf{b}_n \in \mathbf{B}, a(n) \text{ scalar, all } 1 \leq n \leq N. \right\}. \tag{2.19}$$

It is easy to verify from the vector space axioms that $\operatorname{span} \mathbf{B} \subset \mathbf{X}$. This definition agrees with Equation 2.5 for finite sets $\mathbf{B}$, and yields all *finite* linear combinations of elements of $\mathbf{B}$ in the general case. Notice that $\operatorname{span} \mathbf{X} = \mathbf{X}$.

Different norms for an infinite dimensional $\mathbf{X}$ need not be comparable, so a fixed norm must be chosen before the Schauder bases axioms can be verified. Also, the ordering of Schauder basis vectors is important in the infinite dimensional case since the same vectors taken in another order may lose the completeness property. If a

set of vectors is a Schauder basis in every order, then it is called an *unconditional Schauder basis.*

Any subset of a Schauder basis is linearly independent and has the unique representation property. However, proper subsets of Schauder bases are never complete.

In an N-dimensional vector space, linear independence implies unique representation, and completeness requires taking no limits: $\|\mathbf{x} - \sum_{n=1}^{N} c(n)\mathbf{b}_n\| = 0$ for any expansion, regardless of norm. Thus, every basis for a finite dimensional vector space is a Schauder basis. But this is not so in infinite dimensions. Consider the following examples, each of which generalizes Euclidean N-space:

1. ℓ^2, the space of square-summable infinite sequences $\mathbf{x} = \{x(k) : k = 1, 2, \ldots\}$, namely, those for which

$$\sum_{n=1}^{\infty} |x(n)|^2 \stackrel{\text{def}}{=} \lim_{N\to\infty} \sum_{n=1}^{N} |x(n)|^2$$

is a finite number. We use the norm

$$\|\mathbf{x}\| \stackrel{\text{def}}{=} \sqrt{\sum_{n=1}^{\infty} |x(n)|^2} \tag{2.20}$$

For example, $x(k) = \frac{1}{k}$ belongs to ℓ^2, with $\|\mathbf{x}\| = \frac{\pi}{\sqrt{6}}$. Addition is term by term, and the constant zero sequence $\mathbf{0} \stackrel{\text{def}}{=} \{0, 0, \ldots\}$ is the additive identity. The standard basis vectors in this space are sequences $\mathbf{e}_n$ whose terms are all zero except for $e_n(n) = 1$. It is easy to prove that for any positive integer N, the vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N$ are linearly independent. It is not much harder to show that $\{\mathbf{e}_n\}$ is in fact a Schauder basis for ℓ^2. However, the set $\{\mathbf{e}_1; \mathbf{g}_1, \mathbf{g}_2, \ldots\}$ defined by

$$\mathbf{g}_n = \frac{1}{n}\mathbf{e}_n - \frac{1}{n+1}\mathbf{e}_{n+1}, \qquad n = 1, 2, \ldots,$$

is linearly independent and complete but does not satisfy unique representation: $\mathbf{e}_1 = \mathbf{g}_1 + \mathbf{g}_2 + \cdots$.

2. **Poly**, the space of polynomial functions defined on the interval $[-1, 1]$, such as $p(x) = x^4 - \pi x^2 + \frac{3}{7}$. Addition is term by term, and the constant zero polynomial $\mathbf{0}$ is the additive identity. The monomials $1, x, x^2, \ldots, x^n$ are linearly independent for any n, and form an unconditional Schauder basis in the norm

$$\|a(0) + a(1)x + \cdots + a(N)x^N\| \stackrel{\text{def}}{=} \sqrt{|a(0)|^2 + \cdots + |a(N)|^2}. \tag{2.21}$$

It is left as an exercise for the reader to find a linearly independent and complete set of polynomials that nonetheless fails to be a Schauder basis with respect to this norm.

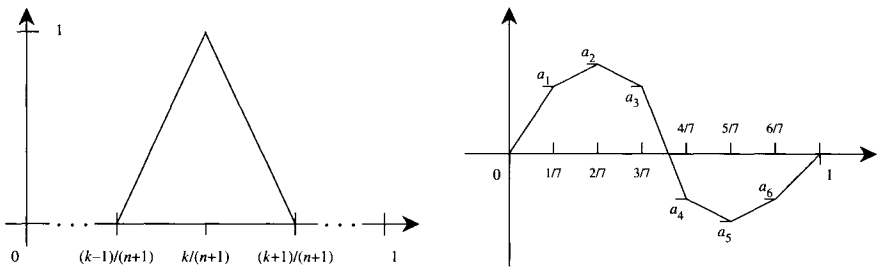


Figure 2.2: Left: Graph of the hat function h_k. Right: Polygon graph of the piecewise linear continuous function $\sum_{k=1}^{n} a(k)h_k$, defined on $[0,1]$, for $n = 6$.

3. **Lip**, the space of complex-valued continuous functions $f = f(t)$, defined on the interval $[0,1]$, that satisfy a *Lipschitz condition*: there is some number C such that $|f(s) - f(t)| \leq C|s - t|$ for all $s, t \in [0,1]$. Vectors in this space are added pointwise: $[f + g](t) = f(t) + g(t)$. The constant zero function, $f(t) = 0$ for all t, is the additive identity.

   There is no maximal finite set of linearly independent functions in this space. Given any integer $n > 0$, there are n continuous *hat functions* $h_1, \ldots, h_n$, as depicted at left in Figure 2.2, defined by

$$h_k(t) = \begin{cases} 0, & \text{if } t < \frac{k-1}{n+1} \text{ or } t > \frac{k+1}{n+1}; \\ (n+1)t - (k-1), & \text{if } \frac{k-1}{n+1} \leq t \leq \frac{k}{n+1}; \\ (k+1) - (n+1)t, & \text{if } \frac{k}{n+1} \leq t \leq \frac{k+1}{n+1}. \end{cases} \tag{2.22}$$

   Such functions satisfy a Lipschitz condition with $C = n+1$. The superposition $a(1)h_1 + \cdots + a(n)h_n$ is the piecewise linear function whose graph is the polygon passing through the points $(0,0), \{(\frac{k}{n+1}, a(k)) : k = 1, \ldots, n\}$, and $(1,0)$. An example is depicted at right in Figure 2.2. It is the zero function if and only only if $a(1) = \cdots = a(n) = 0$. Since n can be as large as we like, **Lip** cannot be finite dimensional.

   In the norm

$$\|f\| \stackrel{\text{def}}{=} \sqrt{\int_0^1 |f(t)|^2 \, dt}, \tag{2.23}$$

   the functions $\{c_n : n = 0, 1, \ldots\}$ defined by $c_n(t) = \cos(\pi n t)$ are a Schauder basis for **Lip**, although the proof of this fact is beyond our scope.

4. $L^2(\mathbf{R})$, also denoted just L^2, is the space of square-integrable complex-valued functions $f = f(t)$ defined at all $t \in \mathbf{R}$. Vector addition is pointwise, and the constant zero function is the additive identity, as in **Lip**. The norm is similar to the one in **Lip**:

$$\|f\| \stackrel{\text{def}}{=} \sqrt{\int_{\mathbf{R}} |f(t)|^2 \, dt}, \tag{2.24}$$

and a function f belongs to $L^2(\mathbf{R})$ if and only if $\|f\|$ is finite. Thus the nonzero constant functions, and more generally the nonzero polynomials, are not members of $L^2(\mathbf{R})$. Since $\|f\|^2$ is called the *energy* of a function, L^2 is sometimes called the space of finite-energy signals.

L^2, like **Lip**, is infinite dimensional. Since there is no continuity assumption, we may build a simple set of basis functions from the *indicator function* of the unit interval $[0, 1)$:

$$\mathbf{1}(t) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } 0 \le t < 1; \\ 0, & \text{if } t < 0 \text{ or } t \ge 1. \end{cases} \tag{2.25}$$

Given any integer k, put $e_k(t) \stackrel{\text{def}}{=} \mathbf{1}(t-k)$ to get the characteristic function of $[k, k+1)$. The functions $\{e_k : k \in \mathbf{Z}\}$ are clearly linearly independent in $L^2(\mathbf{R})$, and there are infinitely many of them. We can also introduce a *scale* index j and put $e_{jk}(t) \stackrel{\text{def}}{=} 2^{-j/2}\mathbf{1}(2^{-j}t-k)$, which is normalized to guarantee $\|e_{jk}\| = 1$. The set $\{e_{jk} : j, k \in \mathbf{Z}\}$ is dense in L^2, but it is clearly not linearly independent. However, the fixed-scale functions $E_j = \{e_{jk} : k \in \mathbf{Z}\}$ are linearly independent, and given a function $f \in L^2(\mathbf{R})$ and $\epsilon > 0$, we can find a scale J and a function $f_J \in \operatorname{span} E_J \subset L^2(\mathbf{R})$ satisfying $\|f - f_J\| < \epsilon$.

2.1.3 Inner product spaces

An *inner product space* $\mathbf{X}$ is a special kind of vector space in which there is also an *inner product*. This is a scalar-valued function on pairs of vectors $\mathbf{u}, \mathbf{v} \in \mathbf{X}$, denoted by $\langle \mathbf{u}, \mathbf{v} \rangle$, that must satisfy the following:

Inner Product Axioms

Hermitean symmetry: For any $\mathbf{u}, \mathbf{v} \in \mathbf{X}$, $\langle \mathbf{u}, \mathbf{v} \rangle = \overline{\langle \mathbf{v}, \mathbf{u} \rangle}$. If all coordinates and scalars are real numbers, this reduces to the ordinary symmetry condition $\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{v}, \mathbf{u} \rangle$;

Nondegeneracy: If $\mathbf{u} \in \mathbf{X}$ and $\mathbf{u} \neq \mathbf{0}$, then $\langle \mathbf{u}, \mathbf{u} \rangle \neq 0$. Equivalently, if $\mathbf{u} \in \mathbf{X}$ and $\langle \mathbf{u}, \mathbf{v} \rangle = 0$ for all $\mathbf{v} \in \mathbf{X}$, then $\mathbf{u} = \mathbf{0}$;

Linearity: For any $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbf{X}$ and any scalars c, d, $\langle \mathbf{u}, c\mathbf{v} + d\mathbf{w} \rangle = c \langle \mathbf{u}, \mathbf{v} \rangle + d \langle \mathbf{u}, \mathbf{w} \rangle$.

Hermitean symmetry implies that $\langle \mathbf{u}, \mathbf{u} \rangle$ is a nonnegative real number; nondegeneracy implies further that this real number is positive whenever $\mathbf{u} \neq \mathbf{0}$. Of course $\langle \mathbf{0}, \mathbf{0} \rangle = 0$. This allows us to define a nondegenerate *derived norm* by the formula $\|\mathbf{u}\| \stackrel{\text{def}}{=} \sqrt{\langle \mathbf{u}, \mathbf{u} \rangle}$, just as in Euclidean N-space.

By linearity and Hermitean symmetry, $\langle c\mathbf{v} + d\mathbf{w}, \mathbf{u} \rangle = \bar{c} \langle \mathbf{v}, \mathbf{u} \rangle + \bar{d} \langle \mathbf{w}, \mathbf{u} \rangle$. Thus $\langle c\mathbf{u}, c\mathbf{u} \rangle = |c|^2 \langle \mathbf{u}, \mathbf{u} \rangle$, so the derived norm satisfies $\|c\mathbf{u}\| = |c| \|\mathbf{u}\|$. We will see in Lemma 2.4 that the other sublinearity condition also holds, so a derived norm indeed satisfies the norm axioms.

If all scalars and coordinates are real numbers, the inner product is real-valued and linear in the first factor as well: $\langle c\mathbf{v} + d\mathbf{w}, \mathbf{u} \rangle = c \langle \mathbf{v}, \mathbf{u} \rangle + d \langle \mathbf{w}, \mathbf{u} \rangle$.

Euclidean N-space, whether real or complex, is an example of a finite dimensional inner product space. Any proof using just the abstract inner product axioms works for every N at once because no dimension N is specified. There are infinite dimensional inner product spaces, too, such as the three examples introduced previously:

1. The vector space ℓ^2 has an inner product

$$\langle \mathbf{x}, \mathbf{y} \rangle \stackrel{\text{def}}{=} \sum_{k=1}^{\infty} \overline{x(k)} y(k). \tag{2.26}$$

   The derived norm is the same as the one in Equation 2.20.

2. **Poly** may be equipped with the following inner product:

$$\langle \mathbf{p}, \mathbf{q} \rangle \stackrel{\text{def}}{=} \int_{-1}^{1} \frac{\overline{p(x)}\, q(x)}{\sqrt{1-x^2}}\, dt, \tag{2.27}$$

   where the integral is interpreted as the limit of integrals on $[-1+\epsilon, 1-\epsilon]$ as $\epsilon \to 0+$. The derived norm is different from, and not comparable to, the one in Equation 2.21. To see why, consider $p(x) = x^n$. Then the Equation 2.21 norm is $\|\mathbf{p}\| = 1$ for all n, whereas the change of variable $x \leftarrow \sin\theta$ yields

$$\langle \mathbf{p}, \mathbf{p} \rangle = \int_{-1}^{1} \frac{x^{2n}}{\sqrt{1-x^2}}\, dx = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \sin^{2n}\theta\, d\theta = \frac{(2n-1)!!}{(2n)!!}\pi = \frac{(2n)!}{2^{2n}(n!)^2}\pi,$$

   as evaluated in Zwillinger's *Standard Mathematical Tables and Formulae*, page 399, Equation 633. But Stirling's asymptotic formula $n! \sim \sqrt{2\pi n}(n/e)^n$ as $n \to \infty$, which is also in Zwillinger on page 496, Section 6.11.7, gives the approximation $\langle \mathbf{p}, \mathbf{p} \rangle \sim \sqrt{\pi/n}$ as $n \to \infty$, so the derived norm of x^n tends to zero as $n \to \infty$.

3. The vector space **Lip** has an inner product

$$\langle \mathbf{u}, \mathbf{v} \rangle \stackrel{\text{def}}{=} \int_{0}^{1} \overline{u(t)}\, v(t)\, dt, \tag{2.28}$$

   for $\mathbf{u} = u(t)$ and $\mathbf{v} = v(t)$. It is left as an exercise to prove that this inner product is Hermitean symmetric, nondegenerate, and linear. The derived norm is the same as the one in Equation 2.23.

4. The vector space $L^2(\mathbf{R})$ has an inner product

$$\langle \mathbf{u}, \mathbf{v} \rangle \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} \overline{u(t)}\, v(t)\, dt, \tag{2.29}$$

   for $\mathbf{u} = u(t)$ and $\mathbf{v} = v(t)$. The derived norm is the same as the one in Equation 2.24.

There may be a choice of inner products in a vector space. For example, $\langle \mathbf{p}, \mathbf{q} \rangle = \int_{-1}^{1} \overline{p(x)} q(x)\, dx$ is an alternative inner product for **Poly**, and $\langle \mathbf{x}, \mathbf{y} \rangle = x_1 y_1 + 17 x_2 y_2$ is another inner product in $\mathbf{E}^2$. The inner product must be specified when the space is defined, but we will use the original choices unless we say otherwise.

Two important inequalities hold for all inner product spaces, including L^2, **Lip**, **Poly**, and ℓ^2 as well as all finite dimensional Euclidean spaces:

Lemma 2.4 *Suppose* $\mathbf{X}$ *is an inner product space with inner product* $\langle \cdot, \cdot \rangle$ *and derived norm* $\| \cdot \|$. *Then we have the following inequalities for all vectors* $\mathbf{u}, \mathbf{v} \in \mathbf{X}$:

Cauchy–Schwarz: $| \langle \mathbf{u}, \mathbf{v} \rangle | \leq \|\mathbf{u}\| \, \|\mathbf{v}\|$;

Minkowski: $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$. *(This is also called the triangle inequality.)*

Proof: For the Cauchy–Schwarz inequality: define a nonnegative real-valued function on real or complex scalars c by

$$g(c) \stackrel{\text{def}}{=} \|\mathbf{u} - c\mathbf{v}\|^2 = \langle \mathbf{u} - c\mathbf{v}, \mathbf{u} - c\mathbf{v} \rangle = \|\mathbf{u}\|^2 - c \langle \mathbf{u}, \mathbf{v} \rangle - \bar{c} \langle \mathbf{v}, \mathbf{u} \rangle + c\bar{c} \|\mathbf{v}\|^2.$$

The inequality holds, with both sides zero, if $\mathbf{v} = \mathbf{0}$. Otherwise, let $c = \overline{\langle \mathbf{u}, \mathbf{v} \rangle} / \|\mathbf{v}\|^2$, which is a critical point $g'(c) = 0$ identifying the minimum of g. Since $g(c) \geq 0$ even at this c, we compute

$$0 \leq \|\mathbf{u}\|^2 - \frac{2 | \langle \mathbf{u}, \mathbf{v} \rangle |^2}{\|\mathbf{v}\|^2} + \frac{| \langle \mathbf{u}, \mathbf{v} \rangle |^2}{\|\mathbf{v}\|^2} \qquad \Rightarrow \quad | \langle \mathbf{u}, \mathbf{v} \rangle |^2 \leq \|\mathbf{u}\|^2 \|\mathbf{v}\|^2.$$

Taking square roots on both sides gives the Cauchy–Schwarz inequality.

For Minkowski's inequality, note that for any real or complex scalar c, $c + \bar{c} \leq 2|c|$. Thus,

$$\begin{aligned} \|\mathbf{u} + \mathbf{v}\|^2 &= \|\mathbf{u}\|^2 + \langle \mathbf{u}, \mathbf{v} \rangle + \overline{\langle \mathbf{u}, \mathbf{v} \rangle} + \|\mathbf{v}\|^2 \\ &\leq \|\mathbf{u}\|^2 + 2 | \langle \mathbf{u}, \mathbf{v} \rangle | + \|\mathbf{v}\|^2 \\ &\leq \|\mathbf{u}\|^2 + 2\|\mathbf{u}\|\|\mathbf{v}\| + \|\mathbf{v}\|^2 \qquad = (\|\mathbf{u}\| + \|\mathbf{v}\|)^2. \end{aligned}$$

Take square roots again to complete the proof. □

In a real inner product space, the Cauchy–Schwarz inequality guarantees that $-1 \leq \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\| \|\mathbf{v}\|} \leq 1$, so that an angle θ between nonzero vectors $\mathbf{u}$ and $\mathbf{v}$ may be defined by

$$\cos\theta \stackrel{\text{def}}{=} \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\| \, \|\mathbf{v}\|} \iff \|\mathbf{u} + \mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2 + 2\|\mathbf{u}\| \, \|\mathbf{v}\| \cos\theta,$$

generalizing the Law of Cosines from two-dimensional Euclidean space.

Orthonormal bases

In any inner product space, vectors $\mathbf{u}, \mathbf{v}$ are said to be *orthogonal* if $\langle \mathbf{u}, \mathbf{v} \rangle = 0$. The zero vector is orthogonal to all other vectors, and no vector except $\mathbf{0}$ is orthogonal to itself.

If $\mathbf{Y}$ is any subset of an inner product space $\mathbf{X}$, then its *orthogonal complement* in $\mathbf{X}$ is a subspace, denoted $\mathbf{Y}^\perp$ and defined as follows:

$$\mathbf{Y}^\perp \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbf{X} : \langle \mathbf{x}, \mathbf{y} \rangle = 0 \text{ for all } \mathbf{y} \in \mathbf{Y}.\} \tag{2.30}$$

For example, $\mathbf{X}^\perp = \{\mathbf{0}\}$ and $\{\mathbf{0}\}^\perp = \mathbf{X}$. Also, if $1 \leq m < N$ and we let $\mathbf{Y} = \operatorname{span}\{\mathbf{e}_1, \ldots, \mathbf{e}_m\} \subset \mathbf{E}^N$, then $\mathbf{Y}^\perp = \operatorname{span}\{\mathbf{e}_{m+1}, \ldots, \mathbf{e}_N\}$. It is left as an exercise to prove the following facts:

Lemma 2.5 *Suppose* $\mathbf{Y}$ *is a subset of an inner product space. Then* $\mathbf{Y} \cap \mathbf{Y}^\perp = \{\mathbf{0}\}$, *and* $\mathbf{Y} \subset (\mathbf{Y}^\perp)^\perp$. □

Lemma 2.6 *Suppose that* $\mathbf{Y} = \operatorname{span}\{\mathbf{y}_n : n = 1, \ldots, N\}$. *If* $\langle \mathbf{x}, \mathbf{y}_n \rangle = 0$ *for all* n, *then* $\mathbf{x} \in \mathbf{Y}^\perp$. □

A basis $\{\mathbf{b}_n\}$ in an inner product space is called an *orthogonal basis* if the vectors are pairwise orthogonal, that is, if $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = 0$ when $i \neq j$, and an *orthonormal basis* if the vectors are pairwise orthogonal and also have unit length: $\langle \mathbf{b}_i, \mathbf{b}_i \rangle = 1$ for all i. These two conditions are summed up with the *Kronecker symbol* $\delta(i - j)$, defined by

$$\{\mathbf{b}_n\} \text{ is orthonormal} \iff \langle \mathbf{b}_i, \mathbf{b}_j \rangle = \delta(i - j) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } i = j; \\ 0, & \text{otherwise.} \end{cases} \tag{2.31}$$

The vectors $\{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$ defined previously form an orthonormal basis for $\mathbf{E}^N$. Any subset of an orthogonal or orthonormal basis inherits orthogonality and orthonormality, respectively.

Linearly independent vectors can be orthonormalized:

Theorem 2.7 (Gram-Schmidt) *Suppose* $\mathbf{X}$ *is an inner product space and* $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_N\} \subset \mathbf{X}$ *is a set of* N *linearly independent vectors. Then there is an orthonormal set* $\mathbf{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_N\} \subset \mathbf{X}$ *with* $\operatorname{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_k\} = \operatorname{span}\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ *for all* $k = 1, \ldots, N$. *In particular,* $\operatorname{span}\mathbf{A} = \operatorname{span}\mathbf{B}$.

Proof: First note that the dimension of $\mathbf{X}$ is at least N, and that $\mathbf{b}_1 \neq \mathbf{0}$. We construct $\mathbf{a}_k$ from $\mathbf{b}_k$ inductively, starting with $\mathbf{a}_1 \stackrel{\text{def}}{=} \frac{1}{\|\mathbf{b}_1\|}\mathbf{b}_1$. Then $\|\mathbf{a}_1\| = 1$, $\operatorname{span}\{\mathbf{a}_1\} = \operatorname{span}\{\mathbf{b}_1\}$ since the two vectors are proportional, and the single-vector $\{\mathbf{a}_1\}$ is an orthogonal set. Now suppose that we have constructed an orthonormal set $\mathbf{A}_k \stackrel{\text{def}}{=} \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$ with the same span as $\mathbf{B}_k \stackrel{\text{def}}{=} \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$. Then the vector

$$\mathbf{a}'_{k+1} \stackrel{\text{def}}{=} \mathbf{b}_{k+1} - \sum_{i=1}^{k} \langle \mathbf{a}_i, \mathbf{b}_{k+1} \rangle \mathbf{a}_i$$

cannot be zero since that would imply $\mathbf{b}_{k+1} \in \operatorname{span} \mathbf{A}_k = \operatorname{span} \mathbf{B}_k$, contradicting the linear independence of $\mathbf{B}$. Thus we may put $\mathbf{a}_{k+1} = \frac{1}{\|\mathbf{a}'_{k+1}\|}\mathbf{a}'_{k+1}$. But then $\mathbf{A}_{k+1} \stackrel{\text{def}}{=} \{\mathbf{a}_1, \ldots, \mathbf{a}_{k+1}\}$ is an orthonormal set since $\mathbf{A}_k$ is orthonormal and

$$\langle \mathbf{a}_j, \mathbf{a}_{k+1} \rangle = \langle \mathbf{a}_j, \mathbf{b}_{k+1} \rangle - \sum_{i=1}^{k} \langle \mathbf{a}_i, \mathbf{b}_{k+1} \rangle \langle \mathbf{a}_j, \mathbf{a}_i \rangle = \langle \mathbf{a}_j, \mathbf{b}_{k+1} \rangle - \langle \mathbf{a}_j, \mathbf{b}_{k+1} \rangle = 0,$$

for $j = 1, \ldots, k$. Finally, $\operatorname{span} \mathbf{A}_{k+1} = \operatorname{span} \mathbf{B}_{k+1}$ since $\mathbf{b}_{k+1} \in \operatorname{span} \mathbf{A}_{k+1}$ and $\mathbf{a}_{k+1} \in \operatorname{span} \mathbf{B}_{k+1}$. □

Note that for all n, $\mathbf{a}_n \in \operatorname{span}\{\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}\}^\perp$, since $\mathbf{a}_n \perp \operatorname{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_{n-1}\}$.

An important consequence of Theorem 2.7 is:

Corollary 2.8 *Every finite dimensional inner product space has an orthonormal basis.* □

The Gram-Schmidt algorithm works on infinite sets, too. If $\mathbf{B}$ is a Schauder basis for an infinite dimensional inner product space, then the algorithm produces an infinite sequence $\{\mathbf{a}_n : n = 1, 2, \ldots\}$ of orthonormal vectors.

Orthonormal bases are useful because the derived norm of a vector can be computed from its expansion coefficients in any orthonormal basis:

Lemma 2.9 *Suppose that $\mathbf{X}$ is an N-dimensional inner product space and $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_N\}$ is an orthonormal basis for $\mathbf{X}$. Then any vector $\mathbf{x} \in \mathbf{X}$ may be written uniquely as a linear combination from $\mathbf{B}$, and we have*

$$\mathbf{x} = \sum_{n=1}^{N} x(n)\mathbf{b}_n \quad \Rightarrow \quad \|\mathbf{x}\|^2 = \sum_{n=1}^{N} |x(n)|^2.$$

Proof: Expand, using the linearity of the inner product:

$$\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle = \left\langle \sum_{n=1}^{N} x(n)\mathbf{b}_n, \sum_{m=1}^{N} x(m)\mathbf{b}_m \right\rangle = \sum_{n=1}^{N} \sum_{m=1}^{N} \overline{x(n)} x(m) \langle \mathbf{b}_n, \mathbf{b}_m \rangle .$$

The result now follows from Equation 2.31. □

Two subspaces $\mathbf{Y}, \mathbf{Z}$ of an inner product space $\mathbf{X}$ are said to be *orthogonal subspaces* if every vector in $\mathbf{Y}$ is orthogonal to every vector in $\mathbf{Z}$, namely, if $\langle \mathbf{y}, \mathbf{z} \rangle = 0$ whenever $\mathbf{y} \in \mathbf{Y}$ and $\mathbf{z} \in \mathbf{Z}$. We then write $\mathbf{Y} \perp \mathbf{Z}$. For example, $\mathbf{Y} \perp \mathbf{Y}^\perp$ for every subspace $\mathbf{Y}$. If $\mathbf{Y} \perp \mathbf{Z}$, then $\mathbf{Y} \cap \mathbf{Z} = \{\mathbf{0}\}$, but the converse does not hold: consider $\mathbf{Y} = \operatorname{span}\{(1, 1)\}$ and $\mathbf{Z} = \operatorname{span}\{(1, 0)\}$ in $\mathbf{X} = \mathbf{E}^2$.

Lemma 2.10 *Suppose $\mathbf{X}$ is a finite dimensional inner product space with subspace $\mathbf{Y} \subset \mathbf{X}$. Then any $\mathbf{x} \in \mathbf{X}$ can be written uniquely as $\mathbf{x} = \mathbf{y} + \mathbf{z}$, where $\mathbf{y} \in \mathbf{Y}$ and $\mathbf{z} \in \mathbf{Y}^\perp$.*

Proof: To show existence, let $\mathbf{B} = \{\mathbf{y}_j\} \subset \mathbf{Y} \subset \mathbf{X}$ be an orthonormal basis for $\mathbf{Y}$. Then the vector $\mathbf{y} = \sum_j \langle \mathbf{y}_j, \mathbf{x} \rangle \mathbf{y}_j \in \operatorname{span} \mathbf{B}$ belongs to $\mathbf{Y}$. It remains to show that $\mathbf{z} \stackrel{\text{def}}{=} \mathbf{x} - \mathbf{y}$ belongs to $\mathbf{Y}^\perp$. But for any basis vector $\mathbf{y}_i$, we have

$$\langle \mathbf{y}_i, \mathbf{y} \rangle = \left\langle \mathbf{y}_i, \sum_j \langle \mathbf{y}_j, \mathbf{x} \rangle \mathbf{y}_j \right\rangle = \sum_j \langle \mathbf{y}_j, \mathbf{x} \rangle \langle \mathbf{y}_i, \mathbf{y}_j \rangle = \sum_j \langle \mathbf{y}_j, \mathbf{x} \rangle \delta(i-j) = \langle \mathbf{y}_i, \mathbf{x} \rangle .$$

Therefore, $\langle \mathbf{y}_i, \mathbf{z} \rangle = \langle \mathbf{y}_i, \mathbf{x} - \mathbf{y} \rangle = \langle \mathbf{y}_i, \mathbf{x} \rangle - \langle \mathbf{y}_i, \mathbf{y} \rangle = 0$. Since this is true for all basis vectors $\mathbf{y}_i$ of $\mathbf{Y}$, we conclude from Lemma 2.6 that $\mathbf{z} \in \mathbf{Y}^\perp$.

To prove uniqueness, suppose that $\mathbf{x} = \mathbf{y} + \mathbf{z} = \mathbf{y}' + \mathbf{z}'$ are two such decompositions. Then $\mathbf{y} - \mathbf{y}' = \mathbf{z}' - \mathbf{z} \in \mathbf{Y} \cap \mathbf{Y}^\perp = \{\mathbf{0}\}$ by Lemma 2.5, so $\mathbf{y} = \mathbf{y}'$ and $\mathbf{z} = \mathbf{z}'$. □

Consequently, if $\mathbf{Y}$ is a subspace of the inner product space $\mathbf{X}$, then $\mathbf{Y} = \mathbf{X}$ if and only if $\mathbf{Y}^\perp = \{\mathbf{0}\}$.

A sum of subspaces $\mathbf{Y}, \mathbf{Z}$ of an inner product space $\mathbf{X}$ is called an *orthogonal sum*, or a *direct sum*, if $\mathbf{Y} \perp \mathbf{Z}$, and then it is written $\mathbf{Y} \oplus \mathbf{Z}$. If more than two subspaces $\mathbf{Y}_1, \ldots, \mathbf{Y}_n \subset \mathbf{X}$ are pairwise orthogonal, then their sum is also called direct. If the direct sum is all of $\mathbf{X}$, then we have a *direct sum decomposition* of $\mathbf{X}$ into orthogonal subspaces. An example of such a direct sum decomposition is

$$\mathbf{E}^N = \operatorname{span}\{\mathbf{e}_1\} \oplus \operatorname{span}\{\mathbf{e}_2\} \oplus \cdots \oplus \operatorname{span}\{\mathbf{e}_N\}. \tag{2.32}$$

Recall that $\operatorname{span}\{\mathbf{e}_2\}$, for example, is those N-tuples whose only nonzero coordinate is number 2.

Biorthogonal dual bases

Suppose $\mathbf{X}$ is an N-dimensional inner product space with basis $\mathbf{B} = \{\mathbf{b}_n : n = 1, \ldots, N\}$, not necessarily orthogonal. Then there is another basis $\mathbf{B}' = \{\mathbf{b}'_n : n - 1, \ldots, N\} \subset \mathbf{X}$, called the *biorthogonal dual* to $\mathbf{B}$, satisfying

$$\langle \mathbf{b}_i, \mathbf{b}'_j \rangle = \langle \mathbf{b}'_i, \mathbf{b}_j \rangle = \delta(i-j), \qquad i, j = 1, \ldots, N. \tag{2.33}$$

To construct it, let $\mathbf{X}_k = \operatorname{span}\{\mathbf{b}_n : n \neq k\}$ for each $k = 1, \ldots, N$. Then $\mathbf{X}_k \neq \mathbf{X}$, since $\mathbf{X}$ is N-dimensional, so $\mathbf{X}_k^\perp \neq \{\mathbf{0}\}$. Hence there must be some vector $\mathbf{v}_k \in \mathbf{X}_k^\perp$ for which $\langle \mathbf{b}_k, \mathbf{v}_k \rangle \neq 0$. Let $\mathbf{b}'_k = \frac{1}{\langle \mathbf{b}_k, \mathbf{v}_k \rangle} \mathbf{v}_k$; then $\mathbf{b}'_k \in \mathbf{X}_k^\perp$, and it is easy to check that $\mathbf{B}$ and $\mathbf{B}' = \{\mathbf{b}'_k : k = 1, \ldots, N\}$ satisfy Equation 2.33.

Lemma 2.11 *Every basis* $\mathbf{B} = \{\mathbf{b}_n : n = 1, \ldots, N\}$ *for an* N*-dimensional inner product space has a unique biorthogonal dual basis.*

Proof: We have existence by the construction given previously. Suppose now that $\{\mathbf{b}'_n\}$ and $\{\mathbf{b}''_n\}$ are biorthogonal duals. Then for fixed n we have $\langle \mathbf{b}'_n - \mathbf{b}''_n, \mathbf{b}_m \rangle = 0$ for all $m = 1, \ldots, N$. Since $\mathbf{B}$ is a basis, we conclude that $\mathbf{b}'_n = \mathbf{b}''_n$. □

Consequently, $\mathbf{B} = \mathbf{B}'$ if and only if $\mathbf{B}$ is an orthonormal basis.

Biorthogonal dual bases are used to compute expansion coefficients:

Lemma 2.12 *Suppose that* $\mathbf{B} = \{\mathbf{b}_n : n = 1, \ldots, N\}$ *and* $\mathbf{B}' = \{\mathbf{b}'_n : n = 1, \ldots, N\}$ *are biorthogonal dual bases for an* N*-dimensional inner product space* $\mathbf{X}$*. Then for any* $\mathbf{x} \in \mathbf{X}$*, we have*

$$\mathbf{x} = \sum_{n=1}^{N} \langle \mathbf{b}'_n, \mathbf{x} \rangle \, \mathbf{b}_n = \sum_{n=1}^{N} \langle \mathbf{b}_n, \mathbf{x} \rangle \, \mathbf{b}'_n.$$

In other words, the expansion coefficients of $\mathbf{x}$ *in* $\mathbf{B}$ *are* $\{c(n) = \langle \mathbf{b}'_n, \mathbf{x} \rangle\}$*, and the expansion coefficients of* $\mathbf{x}$ *in* $\mathbf{B}'$ *are* $\{c'(n) = \langle \mathbf{b}_n, \mathbf{x} \rangle\}$*.*

Proof: Let $\mathbf{x} = \sum_{m=1}^{N} c(m)\mathbf{b}_m$ be the expansion of $\mathbf{x}$ in basis $\mathbf{B}$. Then

$$\langle \mathbf{b}'_n, \mathbf{x} \rangle = \sum_{m=1}^{N} c(m) \langle \mathbf{b}'_n, \mathbf{b}_m \rangle = \sum_{m=1}^{N} c(m)\delta(n-m) = c(n).$$

The proof for $c'(n)$ is similar. □

Consequently, we can expand each basis in terms of the other:

$$\mathbf{b}'_m = \sum_{n=1}^{N} \langle \mathbf{b}'_n, \mathbf{b}'_m \rangle \, \mathbf{b}_n; \qquad \mathbf{b}_m = \sum_{n=1}^{N} \langle \mathbf{b}_n, \mathbf{b}_m \rangle \, \mathbf{b}'_n; \qquad \text{for } m = 1, \ldots, N.$$

For example, consider the basis $\{\mathbf{f}_n = \mathbf{e}_1 + \cdots + \mathbf{e}_n : n = 1, \ldots, N\}$ defined by Equation 2.8. It is left as an exercise to check that its biorthogonal dual is $\{\mathbf{f}'_n \stackrel{\text{def}}{=} \mathbf{e}_n - \mathbf{e}_{n+1} : n = 1, \ldots, N\}$, or

$$\mathbf{f}'_1 = \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \ldots, \quad \mathbf{f}'_{N-1} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ -1 \end{pmatrix}, \quad \mathbf{f}'_N = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{pmatrix}, \tag{2.34}$$

where for notational convenience we write $\mathbf{e}_{N+1} \stackrel{\text{def}}{=} \mathbf{0}$. To use Lemma 2.12, we first compute $\langle \mathbf{f}_i, \mathbf{f}_j \rangle = \min\{i, j\}$ and

$$\langle \mathbf{f}'_i, \mathbf{f}'_j \rangle = \begin{cases} 2, & \text{if } i = j; \\ -1, & \text{if } |i-j| = 1; \\ 0, & \text{if } |i-j| > 1. \end{cases}$$

Then, using $\mathbf{f}_0 \stackrel{\text{def}}{=} \mathbf{0}$ and $\mathbf{f}_{N+1} \stackrel{\text{def}}{=} \mathbf{0}$ for notational convenience, we have

$$\begin{aligned} \mathbf{f}'_m &= -\mathbf{f}_{m-1} + 2\mathbf{f}_m - \mathbf{f}_{m+1}, \\ \mathbf{f}_m &= \mathbf{f}'_1 + 2\mathbf{f}'_2 + \cdots + (m-1)\mathbf{f}'_{m-1} + m[\mathbf{f}'_m + \cdots + \mathbf{f}'_N], \end{aligned}$$

for $m = 1, \ldots, N$.

2.2 Linear Transformations

A *linear transformation* is a function $T : \mathbf{X} \to \mathbf{Y}$ between vector spaces $\mathbf{X}$ and $\mathbf{Y}$ that preserves linear combinations: for vectors $\mathbf{u}, \mathbf{v} \in \mathbf{X}$ and scalars a, b, we can expand $T(a\mathbf{u} + b\mathbf{v}) = aT\mathbf{u} + bT\mathbf{v}$. Some examples are:

1. $\mathbf{X} = \mathbf{E}^1$, $\mathbf{Y} = \mathbf{E}^1$; $Tx = 5x$. [Note: $Tx = 5x + 1$ is not a linear transformation.]
2. $\mathbf{X} = \mathbf{E}^1$, $\mathbf{Y} = \mathbf{E}^3$; $Tx = (x, 2x, 3x)$. [Note: $Tx = (x, x^2, x^3)$ is not a linear transformation.]
3. $\mathbf{X} = \mathbf{E}^2$, $\mathbf{Y} = \mathbf{E}^3$; $T(x_1, x_2) = (0, x_1 + x_2, x_1 - x_2)$.
4. $\mathbf{X} = \mathbf{E}^2$, $\mathbf{Y} = \mathbf{E}^2$; $T(x_1, x_2) = (x_1 \cos\theta + x_2 \sin\theta, -x_1 \sin\theta + x_2 \cos\theta)$ for some fixed θ.
5. $\mathbf{X} = \ell^2$, $\mathbf{Y} = \ell^2$; $T(x_1, x_2, x_3, \ldots) = (x_1, 2x_2, 3x_3, \ldots, nx_n, \ldots)$.

Systems of simultaneous linear equations are one concrete source of linear transformations. Suppose that numbers $x(1), \ldots, x(N)$ are sought satisfying the M linear equations

$$\begin{aligned} A(1,1)x(1) + \cdots + A(1,N)x(N) &= b(1), \\ A(2,1)x(1) + \cdots + A(2,N)x(N) &= b(2), \\ &\vdots \\ A(M,1)x(1) + \cdots + A(M,N)x(N) &= b(M), \end{aligned}$$

where $\{A(m,n) : m = 1, \ldots, M; n = 1, \ldots, N\}$ and $\{b(m) : m = 1, \ldots, M\}$ are given scalar parameters. This may be rewritten as the problem of finding a vector $\mathbf{x} = (x(1), \ldots, x(N))$ in $\mathbf{X} = \mathbf{E}^N$ such that $A\mathbf{x} = \mathbf{b}$, where $\mathbf{b} = (b(1), \ldots, b(M))$ is a vector in $\mathbf{Y} = \mathbf{E}^M$, and

$$A \stackrel{\text{def}}{=} \begin{array}{c} \\ \uparrow \\ M \\ \downarrow \end{array} \begin{array}{c} \longleftarrow\ N\ \longrightarrow \\ \begin{pmatrix} A(1,1) & A(1,2) & \ldots & A(1,N) \\ A(2,1) & A(2,2) & \ldots & A(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ A(M,1) & A(M,2) & \ldots & A(M,N) \end{pmatrix} \end{array} \tag{2.35}$$

is an $M \times N$ *matrix*, an array with M rows and N columns that defines a function from $\mathbf{E}^N$ to $\mathbf{E}^M$ by the formula

$$\begin{aligned} A\mathbf{x} &= \begin{pmatrix} A(1,1) & \cdots & A(1,N) \\ \vdots & \ddots & \vdots \\ A(M,1) & \cdots & A(M,N) \end{pmatrix} \begin{pmatrix} x(1) \\ x(2) \\ \vdots \\ x(N) \end{pmatrix} \\ &\stackrel{\text{def}}{=} \begin{pmatrix} A(1,1)x(1) + \cdots + A(1,N)x(N) \\ \vdots \\ A(M,1)x(1) + \cdots + A(M,N)x(N) \end{pmatrix}. \end{aligned} \tag{2.36}$$

Namely, the m-th coordinate $b(m)$ of $\mathbf{b} = A\mathbf{x}$ is $b(m) = \sum_{n=1}^{N} A(m,n)x(n)$. It is not hard to verify that A is a linear transformation.

2.2.1 Matrix algebra

$M \times N$ matrices can be added componentwise:

$$A + B = C \iff C(m,n) = A(m,n) + B(m,n), \quad 1 \le m \le M, 1 \le n \le N.$$

They can also be multiplied by scalars: $cA = \{cA(m,n)\}$. Thus they form a vector space, which is denoted $\mathbf{Mat}(M \times N)$, satisfying the axioms of Section 2.1.2. This space is NM dimensional, with one basis being the *elementary matrices* $\{\mathbf{e}_{ij} : 1 \le i \le M; 1 \le j \le N\} \subset \mathbf{Mat}(M \times N)$, defined by

$$\mathbf{e}_{ij}(m,n) = \delta(i-m)\delta(j-n) = \begin{cases} 1, & \text{if } m = i \text{ and } n = j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.37}$$

The inner product

$$\langle A, B \rangle \stackrel{\text{def}}{=} \sum_{m=1}^{M} \sum_{n=1}^{N} \overline{A(m,n)}\, B(m,n) \tag{2.38}$$

satisfies the inner product axioms, and makes $\mathbf{Mat}(M \times N)$ an NM-dimensional inner product space. The elementary matrices are in fact an orthonormal basis in this space.

Likewise, linear transformations $T : \mathbf{X} \to \mathbf{Y}$ can be added together and multiplied by scalars to give other linear transformations, so they too form a vector space which we shall denote by $\mathbf{Lin}(\mathbf{X}, \mathbf{Y})$. When $\mathbf{X}$ and $\mathbf{Y}$ are N-dimensional and M-dimensional vector spaces, respectively, then every choice of bases for $\mathbf{X}, \mathbf{Y}$ defines a one-to-one correspondence between $\mathbf{Lin}(\mathbf{X}, \mathbf{Y})$ and $\mathbf{Mat}(M \times N)$:

Theorem 2.13 *Let $T : \mathbf{X} \to \mathbf{Y}$ be a linear transformation between finite dimensional vector spaces $\mathbf{X}$ and $\mathbf{Y}$. Let $\mathbf{P} = \{\mathbf{p}_n : n = 1, \ldots, N\}$ and $\mathbf{Q} = \{\mathbf{q}_m : m = 1, \ldots, M\}$ be bases for $\mathbf{X}$ and $\mathbf{Y}$, respectively. Then there is a unique $M \times N$ matrix $\{A(m,n)\}$ representing T with respect to the bases $\mathbf{P}, \mathbf{Q}$, in the following sense:*

1. *For any $\mathbf{x} = \sum_{n=1}^{N} x(n)\mathbf{p}_n \in \mathbf{X}$ we can compute $T\mathbf{y} = \sum_{m=1}^{M} y(m)\mathbf{q}_m \in \mathbf{Y}$ by the formula*

$$y(m) = \sum_{n=1}^{N} A(m,n)x(n).$$

2. *If any other matrix $\{A'(m,n)\}$ satisfies*

$$\sum_{n=1}^{N} A'(m,n)x(n) = \sum_{n=1}^{N} A(m,n)x(n),$$

for all $\mathbf{x} \in \mathbf{X}$ and all $m = 1, \ldots, M$, then $A'(m,n) = A(m,n)$ for all $n = 1, \ldots, N$ and all $m = 1, \ldots, M$.

Proof: Using linearity, if $\mathbf{x} \in \mathbf{X}$ has the expansion $\mathbf{x} = \sum_{n=1}^{N} x(n)\mathbf{p}_n$, then $T\mathbf{x} \in \mathbf{Y}$ has the expansion

$$T\mathbf{x} = \sum_{n=1}^{N} x(n) T\mathbf{p}_n = \sum_{n=1}^{N} x(n) \sum_{m=1}^{M} A(m,n)\mathbf{q}_m = \sum_{m=1}^{M} \left(\sum_{n=1}^{N} x(n) A(m,n) \right) \mathbf{q}_m,$$

where for each $n = 1, \ldots, N$, the *matrix coefficients* $\{A(m,n) : m = 1, \ldots, M\}$ give the expansion of $T\mathbf{p}_n$ in the basis $\mathbf{Q}$. We may therefore compute the expansion coefficients $\{y(m) : m = 1, \ldots, M\}$ of $T\mathbf{x} = \sum_{m=1}^{M} y(m)\mathbf{q}_m$ by the formula $y(m) = \sum_{n=1}^{N} A(m,n)x(n)$.

For the second point, use $\mathbf{x} = \mathbf{e}_n$ to extract column n of the matrices, for fixed $n \in \{1, \ldots, N\}$. Then $A'(m,n) = A(m,n)$ for all $m = 1, \ldots, M$. Since n was arbitrary, the two matrices must be equal. □

Thus every linear transformation T between finite dimensional vector spaces can be represented by a matrix A whose matrix coefficients depend on the bases $\mathbf{P}, \mathbf{Q}$, so the correspondence may be pictured as follows:

$$T \overset{\mathbf{P},\mathbf{Q}}{\longleftrightarrow} A \tag{2.39}$$

Using the standard bases $\{\mathbf{e}_n\}$ from Equations 2.7, the previously-defined examples of linear transformations have the following matrices:

1. $Tx = 5x$ has the 1×1 matrix $A = (5)$ with respect to the standard bases $\mathbf{P} = \mathbf{Q} = \{\mathbf{e}_1\}$;
2. $T\mathbf{x} = (1x, 2x, 3x)$ has the 3×1 matrix $A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ with respect to the standard bases $\mathbf{P} = \{\mathbf{e}_1\}$ and $\mathbf{Q} = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$;
3. $T(x_1, x_2) = (0, x_1 + x_2, x_1 - x_2)$ has the 3×2 matrix $A = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & -1 \end{pmatrix}$ with respect to the standard bases $\mathbf{P} = \{\mathbf{e}_1, \mathbf{e}_2\}$ and $\mathbf{Q} = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$;
4. $T(x_1, x_2) = (x_1 \cos\theta + x_2 \sin\theta, -x_1 \sin\theta + x_2 \cos\theta)$ has the 2×2 matrix $A = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$ with respect to the standard bases $\mathbf{P} = \mathbf{Q} = \{\mathbf{e}_1, \mathbf{e}_2\}$.

On the other hand, with respect to the bases $\{\mathbf{f}_n\}$ of Equation 2.8, we get the following matrices:

1. Since $\mathbf{e}_1 = \mathbf{f}_1$, the vector $\mathbf{x} = x\mathbf{e}_1 = x\mathbf{f}_1$, so $T\mathbf{x} = 5x$ has the same 1×1 matrix $A = (5)$ with respect to $\mathbf{P} = \mathbf{Q} = \{\mathbf{f}_1\}$;
2. Again, $\mathbf{x} = x\mathbf{e}_1 = x\mathbf{f}_1$, but the output $T\mathbf{x} = x\mathbf{e}_1 + 2x\mathbf{e}_2 + 3x\mathbf{e}_3 = -x\mathbf{f}_1 - x\mathbf{f}_2 + 3x\mathbf{f}_3$, so the matrix of T with respect to $\mathbf{P} = \{\mathbf{f}_1\}$ and $\mathbf{Q} = \{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ is $A = \begin{pmatrix} -1 \\ -1 \\ 3 \end{pmatrix}$;

3. Note that $\mathbf{x} = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 = (x_1 - x_2)\mathbf{f}_1 + x_2\mathbf{f}_2$, while the output is $(0)\mathbf{e}_1 + (x_1 + x_2)\mathbf{e}_2 + (x_1 - x_2)\mathbf{e}_3 = -(x_1 + x_2)\mathbf{f}_1 + 2x_2\mathbf{f}_2 + (x_1 - x_2)\mathbf{f}_3$. Now,

$$\begin{aligned} -(x_1 + x_2) &= (-1)(x_1 - x_2) + (-2)(x_2), \\ 2x_2 &= (0)(x_1 - x_2) + (2)(x_2), \\ (x_1 - x_2) &= (1)(x_1 - x_2) + (0)(x_2), \end{aligned}$$

so T has the 3×2 matrix $A = \begin{pmatrix} -1 & -2 \\ 0 & 2 \\ 1 & 0 \end{pmatrix}$ with respect to $\mathbf{P} = \{\mathbf{f}_1, \mathbf{f}_2\}$ and $\mathbf{Q} = \{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$.

In example 4, the vectors $\mathbf{g}_1 \stackrel{\text{def}}{=} (\cos\theta, -\sin\theta)$ and $\mathbf{g}_2 \stackrel{\text{def}}{=} (\sin\theta, \cos\theta)$ form an orthonormal basis for $\mathbf{E}^2$ for any fixed θ. Thus the linear transformation $T\mathbf{x} = (x_1\cos\theta + x_2\sin\theta, -x_1\sin\theta + x_2\cos\theta) = x_1\mathbf{g}_1 + x_2\mathbf{g}_2$ has the simple matrix $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ with respect to $\mathbf{P} = \{\mathbf{e}_1, \mathbf{e}_2\}$ and $\mathbf{Q} = \{\mathbf{g}_1, \mathbf{g}_2\}$.

Example 5, $T(x_1, x_2, x_3, \ldots) = (x_1, 2x_2, 3x_3, \ldots, nx_n, \ldots)$, may be said to have an "infinite matrix" with respect to the bases of elementary sequences $\mathbf{P} = \mathbf{Q} = \{\mathbf{e}_n : n = 1, 2, \ldots\}$:

$$A = \begin{pmatrix} 1 & 0 & 0 & \ldots \\ 0 & 2 & 0 & \ldots \\ 0 & 0 & 3 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

Except in such simple cases, though, we cannot perform computations with infinite arrays, and must rely on other formulas to represent linear transformations on infinite dimensional vector spaces.

If $\mathbf{X}$ and $\mathbf{Y}$ are finite dimensional inner product spaces with respective bases $\mathbf{P} = \{\mathbf{p}_n\}$ and $\mathbf{Q} = \{\mathbf{q}_m\}$, then matrix coefficients can be computed using the inner product and the biorthogonal dual $\{\mathbf{q}'_m\}$ of $\{\mathbf{q}_m\}$:

$$A(m, n) = \langle \mathbf{q}'_m, T\mathbf{p}_n \rangle . \tag{2.40}$$

In the special case $\mathbf{X} = \mathbf{Y} = \mathbf{E}^N$ with orthonormal basis $\mathbf{P} = \mathbf{Q} = \{\mathbf{e}_n\}$, the formula simplifies to $A(m, n) = \langle \mathbf{e}_m, T\mathbf{e}_n \rangle$.

Composition

Linear transformations can be composed like any other functions. If $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ are vector spaces, and $T : \mathbf{X} \to \mathbf{Y}$ and $S : \mathbf{Y} \to \mathbf{Z}$ are linear transformations, then their *composition* $S \circ T : \mathbf{X} \to \mathbf{Z}$, which is usually written just ST, is defined by $\mathbf{x} \mapsto ST\mathbf{x} = S(T\mathbf{x})$. This is a linear transformation: for all vectors $\mathbf{x}, \mathbf{y} \in \mathbf{X}$ and all scalars a, b, we have

$$ST(a\mathbf{x} + b\mathbf{y}) = S(aT\mathbf{x} + bT\mathbf{y}) = aST(\mathbf{x}) + bST(\mathbf{y}).$$

For example, the linear transformations of examples 1 and 2 defined previously have composition $x \mapsto (5x, 10x, 15x)$.

The composition of linear transformations on finite dimensional vector spaces defines a *matrix product* of their respective matrices. Suppose that $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ are bases for the spaces $\mathbf{X}$ (of dimension N), $\mathbf{Y}$ (of dimension K), and $\mathbf{Z}$ (of dimension M), respectively, and we make the correspondences

$$S \overset{\mathbf{Q},\mathbf{R}}{\longleftrightarrow} A \in \mathbf{Mat}(M \times K); \qquad T \overset{\mathbf{P},\mathbf{Q}}{\longleftrightarrow} B \in \mathbf{Mat}(K \times N).$$

Then $ST \overset{\mathbf{P},\mathbf{R}}{\longleftrightarrow} AB \in \mathbf{Mat}(M \times N)$, where

$$AB(m,n) \overset{\text{def}}{=} \sum_{k=1}^{K} A(m,k)B(k,n), \qquad 1 \le m \le M; 1 \le n \le N. \tag{2.41}$$

Namely, the m-th coordinate of $AB\mathbf{x}$ is

$$AB\mathbf{x}(m) = \sum_{n=1}^{N} AB(m,n)x(n) = \sum_{n=1}^{N}\sum_{k=1}^{K} A(m,k)B(k,n)x(n).$$

The set $\mathbf{Lin}\,(\mathbf{X}) \overset{\text{def}}{=} \mathbf{Lin}\,(\mathbf{X},\mathbf{X})$ of linear transformations $T : \mathbf{X} \to \mathbf{X}$ from an inner product space $\mathbf{X}$ to itself is preserved by composition. If $\mathbf{X}$ is finite dimensional with basis $\{\mathbf{b}_n : n = 1, \ldots, N\}$ for $\mathbf{X}$, the corresponding set $\mathbf{Mat}(N \times N)$ of *square matrices* is likewise closed under matrix multiplication.

$\mathbf{Lin}\,(\mathbf{X})$, and thus $\mathbf{Mat}(N \times N)$, form a special kind of vector space called an *associative algebra (with 1)*, having the following properties for all vectors $\mathbf{U}, \mathbf{V}, \mathbf{W}$ and scalars s, t:

Associative Algebra (With 1) Axioms

Additive associativity: $\mathbf{U} + (\mathbf{V} + \mathbf{W}) = (\mathbf{U} + \mathbf{V}) + \mathbf{W}$.

Additive commutativity: $\mathbf{U} + \mathbf{V} = \mathbf{V} + \mathbf{U}$.

Additive identity: There is a unique vector $\mathbf{0}$ satisfying $\mathbf{U} + \mathbf{0} = \mathbf{0} + \mathbf{U} = \mathbf{U}$ for all $\mathbf{U}$.

Additive inverses: $\mathbf{U} + (-1)\mathbf{U} = (-1)\mathbf{U} + \mathbf{U} = \mathbf{0}$.

Scalar multiplication distributivity: $t(\mathbf{U} + \mathbf{V}) = t\mathbf{U} + t\mathbf{V}$.

Scalar multiplication associativity I: $(st)\mathbf{U} = s(t\mathbf{U})$.

Multiplicative associativity: $\mathbf{U}(\mathbf{V}\mathbf{W}) = (\mathbf{U}\mathbf{V})\mathbf{W}$.

Multiplicative identity: There is a unique vector $\mathbf{1}$, different from $\mathbf{0}$, satisfying $\mathbf{U}\,\mathbf{1} = \mathbf{1}\,\mathbf{U} = \mathbf{U}$ for all $\mathbf{U}$.

Distributivity: $\mathbf{U}(\mathbf{V} + \mathbf{W}) = \mathbf{U}\mathbf{V} + \mathbf{U}\mathbf{W}$.

Scalar multiplication commutativity: $(t\mathbf{U})\mathbf{V} = \mathbf{U}(t\mathbf{V})$.

Scalar multiplication associativity II: $(t\mathbf{U})\mathbf{V} = t(\mathbf{U}\mathbf{V})$.

Note that the first six of these are just the vector space axioms for the operations of matrix addition and scalar multiplication. In $\mathbf{Lin}(\mathbf{X})$, the additive identity $\mathbf{0}$ is the transformation $\mathbf{x} \mapsto \mathbf{0}$ and the multiplicative identity $\mathbf{1}$ is $\mathbf{x} \mapsto \mathbf{x}$. In $\mathbf{Mat}(N \times N)$, these are respectively the *zero matrix* $0(m,n) = 0$ for all $1 \leq n, m \leq N$, and the $N \times N$ *identity matrix*

$$Id \stackrel{\text{def}}{=} \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}. \tag{2.42}$$

We may also write $Id(m,n) = \delta(m-n)$, where δ is the Kronecker symbol of Equation 2.31, or even

$$Id = \sum_{n=1}^{N} \mathbf{e}_{nn},$$

where each $\mathbf{e}_{nn}$ is one of the elementary matrices defined in Equation 2.37.

It is not true that $AB = BA$ for all $A, B \in \mathbf{Mat}(N \times N)$, nor can we conclude from $AB = \mathbf{0}$ that either $A = \mathbf{0}$ or $B = \mathbf{0}$. Examples are readily obtained from $\mathbf{Mat}(2 \times 2)$:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}; \qquad B = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}.$$

Evidently $AB = \mathbf{0}$, even though $A \neq \mathbf{0}$ and $B \neq \mathbf{0}$. Likewise, $BA = B \neq AB$.

2.2.2 Adjoints and projections

If $\mathbf{X}$ and $\mathbf{Y}$ are inner product spaces, then every linear transformation $T : \mathbf{X} \to \mathbf{Y}$ has an *adjoint linear transformation* $T^* : \mathbf{Y} \to \mathbf{X}$, or simply *adjoint*, defined by the relation

$$\langle \mathbf{x}, T^*\mathbf{y} \rangle = \langle T\mathbf{x}, \mathbf{y} \rangle, \qquad \text{for all } \mathbf{x} \in \mathbf{X} \text{ and all } \mathbf{y} \in \mathbf{Y}.$$

To compute $\mathbf{z} = T^*\mathbf{y}$ for a given $\mathbf{y} \in \mathbf{Y}$, it is necessary to find $\mathbf{z} \in \mathbf{X}$ such that $\langle T\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle$ for all $\mathbf{x} \in \mathbf{X}$. If such a $\mathbf{z}$ exists, it must be unique, for if $\mathbf{z}_1$ and $\mathbf{z}_2$ both satisfy $\langle \mathbf{x}, \mathbf{z}_1 \rangle = \langle \mathbf{x}, \mathbf{z}_2 \rangle = \langle T\mathbf{x}, \mathbf{y} \rangle$ for all $\mathbf{x} \in \mathbf{X}$, then $\langle \mathbf{x}, \mathbf{z}_1 - \mathbf{z}_2 \rangle = 0$ for all $\mathbf{x}$, so $\mathbf{z}_1 = \mathbf{z}_2$ by the nondegeneracy of the $\mathbf{X}$ inner product. The domain of T^* is defined to be the subset of $\mathbf{Y}$ for which such $\mathbf{z}$'s exist.

If the inner product spaces $\mathbf{X}$ and $\mathbf{Y}$ are finite dimensional, then the domain of T^* is all of $\mathbf{Y}$. It follows that $(T^*)^* = T$. The adjoint can then be computed from the matrix representing $\mathbf{T}$:

Lemma 2.14 *Let $\mathbf{X}$ and $\mathbf{Y}$ be finite dimensional inner product spaces. If $T : \mathbf{X} \to \mathbf{Y}$ is given by the matrix $\{A(m,n)\}$ with respect to some fixed bases $\mathbf{P}, \mathbf{Q}$, then $T^* : \mathbf{Y} \to \mathbf{X}$ is given by the* adjoint matrix $\{A^*(n,m) = \overline{A(m,n)} : n = 1, \ldots, N; m = 1, \ldots, M\}$, *with respect to the biorthogonal dual bases $\mathbf{Q}', \mathbf{P}'$.*

Proof: Fix bases $\mathbf{P} = \{\mathbf{p}_n\}$ and $\mathbf{Q} = \{\mathbf{q}_m\}$ for $\mathbf{X}$ and $\mathbf{Y}$, respectively, and let $\mathbf{P}' = \{\mathbf{p}'_n\}, \mathbf{Q}' = \{\mathbf{q}'_m\}$ be their respective biorthogonal duals. Note that the

biorthogonal dual to the basis $\{\mathbf{p}'_n\}$ is $\{\mathbf{p}_n\}$. We can use Equation 2.40 to compute $A^*(n,m)$, the matrix coefficient of T^* with respect to the dual bases:

$$A^*(n,m) = \langle \mathbf{p}_n, T^*\mathbf{q}'_m \rangle = \overline{\langle T^*\mathbf{q}'_m, \mathbf{p}_n \rangle} = \overline{\langle \mathbf{q}'_m, T\mathbf{p}_n \rangle} = \overline{A(m,n)},$$

where the second equality follows from the Hermitean symmetry of the inner product. □

The adjoint matrix is the *transpose* of the original matrix, wherein the i-th row becomes the i-th column and the j-th column becomes the j-th row, with all coefficients replaced by their complex conjugates. If the inner product spaces $\mathbf{X}, \mathbf{Y}$ are real, then the adjoint matrix is simply the transpose.

Lemma 2.15 *If $T : \mathbf{X} \to \mathbf{Y}$ and $S : \mathbf{Y} \to \mathbf{Z}$ are linear transformations on finite dimensional inner product spaces $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, then their composition $ST : \mathbf{X} \to \mathbf{Z}$ has an adjoint $(ST)^* : \mathbf{Z} \to \mathbf{X}$ satisfying $(ST)^* = T^*S^*$.*

Proof: Compute $\langle (ST)^*\mathbf{z}, \mathbf{x} \rangle = \langle \mathbf{z}, ST\mathbf{x} \rangle = \langle S^*\mathbf{z}, T\mathbf{x} \rangle = \langle T^*S^*\mathbf{z}, \mathbf{x} \rangle$. Since this holds for all $\mathbf{x} \in \mathbf{X}$ and all $\mathbf{z} \in \mathbf{Z}$, the result follows from nondegeneracy of the inner product. □

A linear transformation $T : \mathbf{X} \to \mathbf{X}$ is called *selfadjoint* if $T^* = T$. For example, if $\mathbf{X} = \mathbf{E}^N$ and $\{A(m,n) = \langle \mathbf{e}_m, T\mathbf{e}_n \rangle\}$ is the matrix of T with respect to the standard basis of Equation 2.7, then T is selfadjoint if and only if $\overline{A(n,m)} = A(m,n)$.

Selfadjoint linear transformations arise from certain natural compositions:

Lemma 2.16 *Suppose $\mathbf{X}, \mathbf{Y}$ are finite dimensional inner product spaces and $T : \mathbf{X} \to Y$ is a linear transformation with adjoint T^*. Then $T^*T : \mathbf{X} \to \mathbf{X}$ and $TT^* : \mathbf{Y} \to \mathbf{Y}$ are selfadjoint linear transformations.*

Proof: From Lemma 2.15, we have $(T^*T)^* = T^*(T^*)^* = T^*T$ and $(TT^*)^* = (T^*)^*T^* = TT^*$. □

Note that if $T \overset{\mathbf{P},\mathbf{Q}}{\longleftrightarrow} A \in \mathbf{Mat}(M \times N)$ using bases $\mathbf{P}, \mathbf{Q}$, then $T^*T \overset{\mathbf{P},\mathbf{P}'}{\longleftrightarrow} A^*A \in \mathbf{Mat}(N \times N)$ and $TT^* \overset{\mathbf{Q}',\mathbf{Q}}{\longleftrightarrow} AA^* \in \mathbf{Mat}(M \times M)$, where $\mathbf{P}', \mathbf{Q}'$ are the respective biorthogonal dual bases.

Boundedness

A linear transformation $T : \mathbf{X} \to \mathbf{Y}$ on normed vector spaces $\mathbf{X}, \mathbf{Y}$ is said to be *bounded* if there is some *upper bound*, a number c, such that $\|T\mathbf{x}\| \leq c\|\mathbf{x}\|$ for all $\mathbf{x} \in \mathbf{X}$. All the examples except number 5 are bounded linear transformations.

The smallest upper bound c that works is called the *operator norm* of T, and is denoted by $\|T\|_{\text{op}}$. Note that it depends on the norms chosen for $\mathbf{X}$ and $\mathbf{Y}$. It indicates how much T can stretch a vector: For every $\mathbf{x} \in \mathbf{X}$,

$$\|T\mathbf{x}\| \leq \|T\|_{\text{op}} \|\mathbf{x}\|, \tag{2.43}$$

and for every $c < \|T\|_{\text{op}}$ there is some vector $\mathbf{x}_0 \in \mathbf{X}$ for which $\|T\mathbf{x}_0\| > c\|\mathbf{x}_0\|$. The function $\|\cdot\|_{\text{op}}$ is not derived from any inner product, but it satisfies the Norm Axioms on page 29, plus one more:

Extra Operator Norm Axiom

Submultiplicativity: $\|ST\|_{\text{op}} \leq \|S\|_{\text{op}}\|T\|_{\text{op}}$ for any bounded linear transformations S, T that can be composed.

The proof is left as an exercise.

Not all linear transformations are bounded, but all the ones corresponding to matrices are bounded:

Theorem 2.17 *Suppose* $\mathbf{X}$ *and* $\mathbf{Y}$ *are finite dimensional inner product spaces. If* $T : \mathbf{X} \to \mathbf{Y}$ *is a linear transformation, then* T *is bounded with respect to any norms on* $\mathbf{X}, \mathbf{Y}$.

Proof: First consider the derived norms $\|\cdot\|$. Choose any orthonormal bases $\mathbf{P} = \{\mathbf{p}_n\}$ and $\mathbf{Q} = \{\mathbf{q}_m\}$ for N-dimensional $\mathbf{X}$ and M-dimensional $\mathbf{Y}$, respectively, and let $A = \{A(m,n) = \langle \mathbf{q}_m, T\mathbf{p}_n \rangle : \ m = 1, \ldots, M; n = 1, \ldots, N\}$ be the matrix representing T in those bases. By Lemma 2.9, we have $\|\mathbf{x}\|^2 = \sum_n |x(n)|^2$ for any $\mathbf{x} = \sum_n x(n)\mathbf{p}_n \in \mathbf{X}$. Using the Cauchy–Schwarz inequality for $\mathbf{E}^N$, we estimate

$$\begin{aligned} \|T\mathbf{x}\|^2 &= \left\| \sum_{m=1}^{M} \left(\sum_{n=1}^{N} x(n)A(m,n) \right) \mathbf{q}_m \right\|^2 = \sum_{m=1}^{M} \left| \sum_{n=1}^{N} x(n)A(m,n) \right|^2 \\ &\leq \left(\sum_{n=1}^{N} |x(n)|^2 \right) \left(\sum_{m=1}^{M} \sum_{n=1}^{N} |A(m,n)|^2 \right) = s\,\|\mathbf{x}\|^2, \end{aligned}$$

where $s = \sum_{n,m} |A(m,n)|^2 \geq 0$. Thus $\|T\mathbf{x}\| \leq \sqrt{s}\,\|\mathbf{x}\|$, showing boundedness with respect to the derived norms of $\mathbf{X}$ and $\mathbf{Y}$.

Now let $\|\cdot\|_{\mathbf{X}}$ and $\|\cdot\|_{\mathbf{Y}}$ be arbitrary norms for $\mathbf{X}$ and $\mathbf{Y}$. By Theorem 2.3, there are positive constants A and B such that $\|\mathbf{x}\| \leq B\|\mathbf{x}\|_{\mathbf{X}}$ and $\|T\mathbf{x}\|_{\mathbf{Y}} \leq A\|T\mathbf{x}\|$ for all $\mathbf{x} \in \mathbf{X}$. Therefore, $\|T\mathbf{x}\|_{\mathbf{Y}} \leq AB\sqrt{s}\,\|\mathbf{x}\|_{\mathbf{X}}$ for all $\mathbf{x} \in \mathbf{X}$, so T is bounded with respect to those norms as well. □

The upper bound $c = \sqrt{s}$ used in the proof is called the *Hilbert–Schmidt norm* of the matrix A, and is computed directly by:

$$\|A\|_{\text{HS}} \stackrel{\text{def}}{=} \sqrt{\sum_{m=1}^{M} \sum_{n=1}^{N} |A(m,n)|^2}. \tag{2.44}$$

The relation between $\|A\|_{\text{HS}}$ and $\|T\|_{\text{op}}$ depends on the bases and norms. It is particularly simple for the choices made in the proof of Theorem 2.17:

Corollary 2.18 *Suppose* $T \overset{\mathbf{P},\mathbf{Q}}{\longleftrightarrow} A \in \mathbf{Mat}(M \times N)$, *where* $T : \mathbf{X} \to \mathbf{Y}$ *is a linear transformation and* $\mathbf{X}, \mathbf{Y}$ *are inner product spaces with dimension* N, M *and orthonormal bases* $\mathbf{P}$ *and* $\mathbf{Q}$, *respectively. Then*

$$\|T\|_{op} \leq \|A\|_{HS} \leq \sqrt{MN}\,\|T\|_{op}.$$

Proof: On the right, use the Cauchy–Schwarz inequality and Inequality 2.43:

$$\begin{aligned}\|A\|_{\text{HS}}^2 &= \sum_{m,n} |A(m,n)|^2 = \sum_{m,n} |\langle \mathbf{q}_m, T\mathbf{p}_n\rangle|^2 \\ &\leq \sum_{m,n} \|\mathbf{q}_m\|^2 \|T\|_{\text{op}}^2 \|\mathbf{p}_n\|^2 = \|T\|_{\text{op}}^2 \sum_{m,n} 1 = \|T\|_{\text{op}}^2 MN.\end{aligned}$$

On the left, the inequality follows from the proof of Theorem 2.17. □

It is left as an exercise to show that $\|\cdot\|_{\text{HS}}$ is sublinear and nondegenerate and satisfies the product inequality $\|AB\|_{\text{HS}} \leq \|A\|_{\text{HS}}\|B\|_{\text{HS}}$.

Bounded linear transformations $T : \mathbf{X} \to \mathbf{Y}$ form a normed vector space $\mathbf{Bdd}(\mathbf{X}, \mathbf{Y}) \subset \mathbf{Lin}(\mathbf{X}, \mathbf{Y})$, with the norm $\|\cdot\|_{\text{op}}$. Equivalently, matrices $A \in \mathbf{Mat}(M \times N)$ form a normed vector space with $\|\cdot\|_{\text{HS}}$. If $\mathbf{X}$ and $\mathbf{Y}$ are N and M-dimensional, respectively, and respective orthonormal bases $\mathbf{P} \subset \mathbf{X}$ and $\mathbf{Q} \subset \mathbf{Y}$ are chosen, then the correspondence $T \overset{\mathbf{P},\mathbf{Q}}{\longleftrightarrow} A$ can be used to define $\|T\|_{\text{HS}} = \|A\|_{\text{HS}}$ and $\|A\|_{\text{op}} = \|T\|_{\text{op}}$.

Bounded linear transformations $T : \mathbf{X} \to \mathbf{Y}$ are uniformly continuous functions on $\mathbf{X}$: Given $\epsilon > 0$, take any $\delta > 0$ satisfying $\delta\|T\|_{\text{op}} < \epsilon$, where $\|T\|_{\text{op}}$ is the bound computed using the norms $\|\cdot\|_{\mathbf{X}}$ and $\|\cdot\|_{\mathbf{Y}}$. Then for any $\mathbf{u}, \mathbf{v} \in \mathbf{X}$,

$$\|\mathbf{u} - \mathbf{v}\|_{\mathbf{X}} < \delta \Rightarrow \|T\mathbf{u} - T\mathbf{v}\|_{\mathbf{Y}} = \|T(\mathbf{u} - \mathbf{v})\|_{\mathbf{Y}} \leq \|T\|_{\text{op}}\|\mathbf{u} - \mathbf{v}\|_{\mathbf{X}} \leq \delta\|T\|_{\text{op}} < \epsilon.$$

The special case $\mathbf{X} = \mathbf{Y}$ defines the set $\mathbf{Bdd}(\mathbf{X}) \overset{\text{def}}{=} \mathbf{Bdd}(\mathbf{X}, \mathbf{X})$ of bounded linear transformations from an inner product space $\mathbf{X}$ to itself.

Lemma 2.19 *Suppose* $\mathbf{X}$ *is a finite dimensional inner product space. If* $T \in \mathbf{Bdd}(\mathbf{X})$, *then* $T^* \in \mathbf{Bdd}(\mathbf{X})$ *with* $\|T\|_{op} = \|T^*\|_{op}$.

Proof: $|\langle T^*\mathbf{y}, \mathbf{x}\rangle| = |\langle \mathbf{y}, T\mathbf{x}\rangle| \leq \|T\|_{\text{op}}\|\mathbf{y}\|\|\mathbf{x}\|$ for every $\mathbf{x}, \mathbf{y} \in \mathbf{X}$, by the Cauchy–Schwarz inequality. Now fix an arbitrary $\mathbf{y}$ and put $\mathbf{x} = T^*\mathbf{y}$ to conclude that $\|T^*\mathbf{y}\|^2 \leq \|T\|_{\text{op}}\|T^*\|_{\text{op}}\|\mathbf{y}\|^2$. Since $\mathbf{y}$ was arbitrary and $\|T\|_{\text{op}} \geq 0$, this implies that $\|T^*\|_{\text{op}} \leq \|T\|_{\text{op}}$. A similar argument shows that $\|T\|_{\text{op}} \leq \|T^*\|_{\text{op}}$, so in combination these inequalities force $\|T\|_{\text{op}} = \|T^*\|_{\text{op}}$. □

Trace

The *trace* of a square matrix $A \in \mathbf{Mat}(N \times N)$ is the sum of just its coefficients along the *main diagonal*:

$$\operatorname{tr} A \overset{\text{def}}{=} \sum_{n=1}^{N} A(n,n). \tag{2.45}$$

It is evident that $\operatorname{tr}(A + B) = \operatorname{tr} A + \operatorname{tr} B$, $\operatorname{tr}(cA) = c \operatorname{tr} A$, and $\operatorname{tr}(A^*) = \overline{\operatorname{tr} A}$. We also have

$$\operatorname{tr}(AB) = \sum_n \sum_k A(n,k)B(k,n) = \sum_k \sum_n B(k,n)A(n,k) = \operatorname{tr}(BA), \tag{2.46}$$

so $\mathrm{tr}\,(AB - BA) = 0$ even for noncommuting $A, B \in \mathbf{Mat}(N \times N)$.

Even for nonsquare matrices, some products are square:

$$A, B \in \mathbf{Mat}(M \times N) \Rightarrow \begin{cases} B^*A, A^*B \in \mathbf{Mat}(N \times N); \\ BA^*, AB^* \in \mathbf{Mat}(M \times M). \end{cases}$$

We can write the Equation 2.38 inner product $\langle A, B \rangle$ in terms of the matrix adjoint, matrix product, and trace:

$$\langle A, B \rangle = \mathrm{tr}\,(A^*B) = \mathrm{tr}\,(BA^*) = \overline{\mathrm{tr}\,(AB^*)} = \overline{\mathrm{tr}\,(B^*A)} \tag{2.47}$$

The proof is straightforward. As a result, $\|A\|_{\mathrm{HS}} = \sqrt{\mathrm{tr}\,(A^*A)} = \sqrt{\mathrm{tr}\,(AA^*)}$.

Orthogonal projection

Let $\mathbf{X}$ be an inner product space and suppose that $P : \mathbf{X} \to \mathbf{X}$ is a linear transformation. P is called a *projection* if $P^2 = P$, namely, if $P(P\mathbf{x}) = P\mathbf{x}$ for every $\mathbf{x} \in \mathbf{X}$. The range of P, $P\mathbf{X} \stackrel{\mathrm{def}}{=} \{P\mathbf{x} : \mathbf{x} \in \mathbf{X}\}$, is thus preserved by P. Also, $P\mathbf{X}$ is a subspace of $\mathbf{X}$: if $\mathbf{u} = P\mathbf{x} \in P\mathbf{X}$ and $\mathbf{v} = P\mathbf{x} \in P\mathbf{X}$, then $\mathbf{u} + \mathbf{v} = P(\mathbf{x} + \mathbf{y}) \in P\mathbf{X}$.

The name suggests visualizing $P\mathbf{x}$ as the shadow cast by $\mathbf{x}$ on the ground $P\mathbf{X}$, the place of all shadows. The shadow may be longer than $\mathbf{x}$, depending on the angle of the light. For example, the projection on $\mathbf{E}^2$ defined by $P(x_1, x_2) = (0, 5x_1 + x_2)$ lengthens $(1, 0)$ five times: $\|P(1,0)\| = \|(0,5)\| = 5 = 5\|(1,0)\|$.

In the infinite dimensional case, there are projections which can stretch vectors an arbitrary amount. For example, take $\mathbf{X} = \ell^2$ and define $P\mathbf{x}(2^k) = x(2^k) + x(2^k - 1) + \cdots + x(2^k - k + 1)$ for $k = 1, 2, \ldots$, while $P\mathbf{x}(n) = 0$ for all other indices n. That is,

$$P\mathbf{x} = (0, x(2), 0, x(4) + x(3), 0, 0, 0, x(8) + x(7) + \cdots + x(5), 0, \ldots).$$

Clearly $P^2 = P$. However, for $\mathbf{x} = (0, \ldots, 0, 1, 1, \ldots, 1, 0, , \ldots)$, where the ones are at indices $2^m + 1$ through $2^m + 2^m$, we get $\|\mathbf{x}\| = \sqrt{2^m}$ while $\|P\mathbf{x}\| = 2^m$. We can make m as large as we like, so P is an unbounded projection.

A projection P is said to be *orthogonal* if, for every $\mathbf{u} \in \mathbf{X}$, the vector $\mathbf{u} - P\mathbf{u}$ is orthogonal to every vector $P\mathbf{v}$ in $P\mathbf{X}$:

$$\langle P\mathbf{v}, \mathbf{u} - P\mathbf{u} \rangle = 0, \qquad \text{for all } \mathbf{u}, \mathbf{v} \in \mathbf{X}. \tag{2.48}$$

Orthogonal projections P are bounded by Pythagoras' Theorem:

Lemma 2.20 *If P is an orthogonal projection on an inner product space $\mathbf{X}$, then $\|P\mathbf{x}\| \leq \|\mathbf{x}\|$ for all $\mathbf{x} \in \mathbf{X}$.*

Proof: Write $\mathbf{x} = P\mathbf{x} + (\mathbf{x} - P\mathbf{x})$ and compute the length of both sides:

$$\|\mathbf{x}\|^2 = \|P\mathbf{x}\|^2 + \|\mathbf{x} - P\mathbf{x}\|^2 + \langle P\mathbf{x}, \mathbf{x} - P\mathbf{x} \rangle + \langle \mathbf{x} - P\mathbf{x}, P\mathbf{x} \rangle .$$

But the last two terms are zero since $(\mathbf{x}-P\mathbf{x}) \perp P\mathbf{x}$. The inequality follows because $\|\mathbf{x} - P\mathbf{x}\| \geq 0$. □

The inequality is sharp if $P \neq 0$, for then there is some $\mathbf{x}_0 \in \mathbf{X}$ for which $\mathbf{y}_0 \stackrel{\text{def}}{=} P\mathbf{x}_0 \neq 0$, and then $\|P\mathbf{y}_0\| = \|P^2\mathbf{x}_0\| = \|P\mathbf{x}_0\| = \|\mathbf{y}_0\|$. In particular, this means $\|P\|_{\text{op}} = 1$ unless $P = 0$.

There is an easy way to check whether a projection is orthogonal: orthogonal projections are selfadjoint.

Lemma 2.21 *A projection $P : \mathbf{X} \to \mathbf{X}$ is orthogonal if and only if $P^* = P$.*

Proof: If P is an orthogonal projection, then $\langle P\mathbf{v}, \mathbf{u} - P\mathbf{u}\rangle = 0$, so $\langle P\mathbf{v}, \mathbf{u}\rangle = \langle P\mathbf{v}, P\mathbf{u}\rangle = \langle P^*P\mathbf{v}, \mathbf{u}\rangle$. Since this is true for all $\mathbf{u} \in \mathbf{X}$, it must be that $P\mathbf{v} = P^*P\mathbf{v}$ for all $\mathbf{v} \in \mathbf{X}$. But then $P = P^*P$. Since $P^* = (P^*P)^* = P^*P$, this means that $P = P^*$. The converse is evident: if P is a projection and $P = P^*$, then $\langle P\mathbf{v}, \mathbf{u} - P\mathbf{u}\rangle = \langle \mathbf{v}, P\mathbf{u} - P^2\mathbf{u}\rangle = \langle \mathbf{v}, \mathbf{0}\rangle = 0$, so $\mathbf{u} - P\mathbf{u}$ is orthogonal to $P\mathbf{v}$ for any $\mathbf{u}, \mathbf{v}$. □

If $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$ is any finite orthonormal subset of an inner product space $\mathbf{X}$, then we may define

$$P_{\mathbf{Y}}\,\mathbf{x} \stackrel{\text{def}}{=} \sum_{k=1}^{N} \langle \mathbf{y}_k, \mathbf{x}\rangle\, \mathbf{y}_k. \tag{2.49}$$

It is left as an exercise to show that this formula gives an orthogonal projection onto $\operatorname{span}\mathbf{Y}$. For example, the point $(x, y, z) \in \mathbf{R}^3$ is mapped to the point $(x, y, 0) \in \operatorname{span}\{\mathbf{e}_1, \mathbf{e}_2\}$, the xy-plane, by $P\mathbf{x} = \langle \mathbf{e}_1, \mathbf{x}\rangle\, \mathbf{e}_1 + \langle \mathbf{e}_2, \mathbf{x}\rangle\, \mathbf{e}_2$. Another example, with infinite dimensional $\mathbf{X} = \mathbf{Lip}$, is the one-vector set $\mathbf{Y} = \{\mathbf{1}\} \subset L^2(\mathbf{R})$, where $\mathbf{1} = \mathbf{1}(t)$ is the indicator function of the interval $[0, 1)$ as defined in Equation 2.25. The projection $P_{\mathbf{Y}} f(t) = \langle \mathbf{1}, f\rangle\, \mathbf{1}(t)$ gives a multiple of the indicator function, with $\langle \mathbf{1}, f\rangle = \int_0^1 f(t)\, dt$ being the average of f over the interval $[0, 1)$.

The Gram-Schmidt orthonormalization algorithm (Theorem 2.7) is a sequence of such projections: start with $\mathbf{a}_1 = \frac{1}{\|\mathbf{b}_1\|}\mathbf{b}_1$, and then define $\mathbf{a}'_k = \mathbf{b}_k - P_{\operatorname{span}\mathbf{A}_{k-1}}\mathbf{b}_k$ and $\mathbf{a}_k = \frac{1}{\|\mathbf{a}'_k\|}\mathbf{a}'_k$ for $k > 1$.

Perspective projection

Since a projection P is a linear transformation, it maps the line $\mathbf{E} \stackrel{\text{def}}{=} \{\theta\mathbf{x} + (1-\theta)\mathbf{y} : \theta \in \mathbf{R}\}$ to another line $P\mathbf{E} = \{\theta P\mathbf{x} + (1-\theta)P\mathbf{y} : \theta \in \mathbf{R}\}$. Hence, to draw a three-dimensional geometric object consisting of points connected by lines, we project the points to some plane and then connect them with lines.

To draw the same object with *perspective*, we fix the viewpoint at (x_0, y_0, z_0), distinct from any point on the object, and then compute a different projection: the intersections with the xy-plane of rays from the viewpoint through the points of the object, as depicted in Figure 2.3.

Using similar triangles and elementary geometry, we get the formula for the image point $(x', y', 0)$ in the xy-plane, where the object point (x, y, z) is seen from

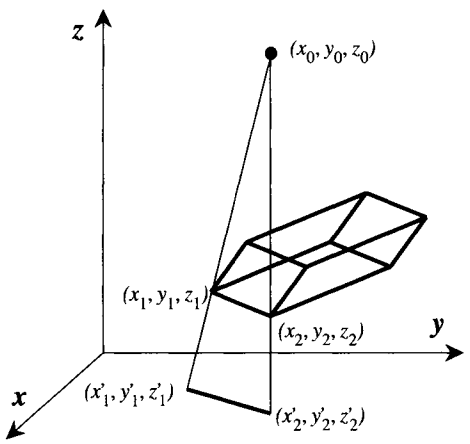


Figure 2.3: Perspective projection from the viewpoint (x_0, y_0, z_0) to the xy-plane.

the viewpoint (x_0, y_0, z_0):

$$\frac{x' - x_0}{z' - z_0} = \frac{x - x_0}{z - z_0}; \qquad \frac{y' - y_0}{z' - z_0} = \frac{y - y_0}{z - z_0}; \qquad z' = 0. \tag{2.50}$$

The transformation $(x, y, z) \mapsto (x', y', z')$ also preserves lines, despite not being a linear transformation, because the line through (x_1, y_1, z_1) and (x_2, y_2, z_2), together with the viewpoint (x_0, y_0, z_0), determine a plane in $\mathbf{R}^3$, and that plane intersects the xy-plane in a line.

2.2.3 Linear independence and invertibility

Suppose that $T : \mathbf{X} \to \mathbf{Y}$ is a linear transformation. Given $\mathbf{y} \in \mathbf{Y}$, there is an $\mathbf{x} \in \mathbf{X}$ solving $T\mathbf{x} = \mathbf{y}$ if and only if $\mathbf{y}$ belongs to the *range* of T, which we may denote by $T\mathbf{X} \stackrel{\text{def}}{=} \{T\mathbf{x} : \mathbf{x} \in \mathbf{X}\}$. In that case, the solution is unique if and only if T is a one-to-one function: $T\mathbf{u} = T\mathbf{v} \iff \mathbf{u} = \mathbf{v}$. By linearity, this is equivalent to the condition that $T\mathbf{x} = \mathbf{0} \iff \mathbf{x} = \mathbf{0}$. We will say that T is *invertible* if for every $\mathbf{y} \in \mathbf{Y}$ there is a unique $\mathbf{x} \in \mathbf{X}$ solving $T\mathbf{x} = \mathbf{y}$. In that case we may define a function $T^{-1} : \mathbf{Y} \to \mathbf{X}$ such that $T(T^{-1}(\mathbf{y})) = \mathbf{y}$ for every $\mathbf{y} \in \mathbf{Y}$ and $T^{-1}(T(\mathbf{x})) = \mathbf{x}$ for every $\mathbf{x} \in \mathbf{X}$. This T^{-1} is a linear transformation since for any $\mathbf{y}_1, \mathbf{y}_2 \in \mathbf{Y}$ and scalars a, b, once we find $\mathbf{x}_1, \mathbf{x}_2 \in \mathbf{X}$ satisfying $T\mathbf{x}_1 = \mathbf{y}_1$ and $T\mathbf{x}_2 = \mathbf{y}_2$, we will have

$$\begin{aligned} T^{-1}(a\mathbf{y}_1 + b\mathbf{y}_2) &= T^{-1}(aT\mathbf{x}_1 + bT\mathbf{x}_2) = T^{-1}(T(a\mathbf{x}_1 + b\mathbf{x}_2)) \\ &= a\mathbf{x}_1 + b\mathbf{x}_2 = aT^{-1}(\mathbf{y}_1) + bT^{-1}(\mathbf{y}_2). \end{aligned}$$

If $\mathbf{X}$ and $\mathbf{Y}$ are finite dimensional vector spaces, with respective dimensions N, M and bases $\mathbf{P} = \{\mathbf{p}_n\}$ and $\mathbf{Q} = \{\mathbf{q}_m\}$, we may explore these existence and

uniqueness issues from the matrix side of the correspondence $T \overset{\mathbf{P},\mathbf{Q}}{\longleftrightarrow} A$, where $A \in \mathbf{Mat}(M \times N)$. For existence, notice that every vector $\mathbf{x} = \sum_n c(n)\mathbf{p}_n \in \mathbf{X}$, expanded in $\mathbf{P}$, determines a coefficient vector $\mathbf{c} = (c(1), \ldots, c(N)) \in \mathbf{E}^N$, and we may compute the expansion coefficients $\mathbf{b} = (b(1), \ldots, b(M)) \in \mathbf{E}^M$ of $T\mathbf{x} = \mathbf{y} = \sum_m b(m)\mathbf{q}_m \in \mathbf{Y}$ by

$$\begin{aligned} \mathbf{b} = A\mathbf{c} &= \begin{pmatrix} A(1,1)c(1) + \cdots + A(1,N)c(N) \\ \vdots \\ A(M,1)c(1) + \cdots + A(M,N)c(N) \end{pmatrix} \\ &= c(1)\begin{pmatrix} A(1,1) \\ \vdots \\ A(M,1) \end{pmatrix} + \cdots + c(N)\begin{pmatrix} A(1,N) \\ \vdots \\ A(M,N) \end{pmatrix}, \end{aligned} \tag{2.51}$$

which is a linear combination of the N vectors $(A(1,1), \ldots, A(M,1))$ through $(A(1,N), \ldots, A(M,N))$ of $\mathbf{E}^M$ formed from columns of A. The set of all such linear combinations is called the range or *column space* of A, and has the form $\{A\mathbf{x} : \mathbf{x} \in \mathbf{E}^N\}$. It is a subspace of $\mathbf{E}^M$, since it is a linear span, and its dimension is no more than the smaller of N and M. Obviously, the linear system of equations $A\mathbf{c} = \mathbf{b}$ of has a solution $\mathbf{c}$ if and only if $\mathbf{b}$ lies in this column space, so if $N < M$, there will certainly be some $\mathbf{b} \in \mathbf{E}^M$ for which $A\mathbf{c} = \mathbf{b}$ has no solution.

For uniqueness, notice that there will be some $\mathbf{c} \neq \mathbf{0}$ for which $A\mathbf{c} = \mathbf{0}$ in Equation 2.51 if and only if the N columns of A are linearly dependent in $\mathbf{E}^M$. They will certainly be linearly dependent if $N > M$, by Theorem 2.2, and $\mathbf{c}$ will give a nonzero $\mathbf{x} \in \mathbf{X}$ for which $T\mathbf{x} = \mathbf{0}$ through the correspondence $\mathbf{X} \overset{\mathbf{P}}{\longleftrightarrow} \mathbf{E}^N$. Hence $T\mathbf{x} = \mathbf{y}$ can have a unique solution $\mathbf{x} \in \mathbf{X}$ for every $\mathbf{y} \in \mathbf{Y}$ only if $\mathbf{X}$ and $\mathbf{Y}$ have the same dimension, which means that $T \overset{\mathbf{P},\mathbf{Q}}{\longleftrightarrow} A$ is a square matrix. But just because its matrix is square doesn't guarantee that a linear transformation is invertible.

On the other side of the correspondence, we will say that $A \in \mathbf{Mat}(N \times N)$ is *invertible* if there is a matrix $B \in \mathbf{Mat}(N \times N)$ satisfying $BA = Id$. Invertibility is equivalent to the columns forming a basis:

Lemma 2.22 *Matrix $A \in \mathbf{Mat}(N \times N)$ is invertible if and only if the columns of A are linearly independent. In that case,*

1. *there is a unique matrix B such that $BA = Id$;*
2. *the columns of B are linearly independent;*
3. *B also satisfies $AB = Id$.*

Proof: By Equation 2.51, any linear combination of the column vectors of A can be expressed as $A\mathbf{c}$ for some column vector $\mathbf{c} = (c(1), \ldots, c(N))$ of expansion coefficients. If A is invertible, then $A\mathbf{c} = \mathbf{0}$ implies $\mathbf{c} = BA\mathbf{c} = \mathbf{0}$, so the columns of A are linearly independent.

Conversely, since the N columns of A are linearly independent, they form a basis for $\mathbf{E}^N$. By Lemma 2.11, this has a unique biorthogonal dual basis, so let A' be the matrix whose columns are the dual vectors. Then $B \stackrel{\text{def}}{=} (A')^*$ satisfies $BA = Id$, so A is invertible.

For 1: the matrix B is unique because the biorthogonal dual basis that determines A' is unique.

For 2: note that any $\mathbf{b} \in \mathbf{E}^N$ can be written as $\mathbf{b} = A\mathbf{c}$ for some $\mathbf{c} \in \mathbf{E}^N$ since the columns of A form a basis. Thus $B\mathbf{b} = \mathbf{0} \Rightarrow \mathbf{c} = BA\mathbf{c} = \mathbf{0} \Rightarrow \mathbf{b} = \mathbf{0}$, so the columns of B are linearly independent.

For 3: note that B is also invertible, since the columns of B are linearly independent. So let $C \in \mathbf{Mat}(N \times N)$ satisfy $CB = Id$. Then by the multiplicative associativity axiom, $AB = (CB)AB = C(BA)B = CB = Id$. □

Since the inverse of an invertible matrix A is unique and works on both sides, it is denoted by A^{-1} and is called the *multiplicative inverse* of A. The proof gives us a construction for the biorthogonal dual to a basis written as a matrix A of column vectors: it is the column vectors of $(A^{-1})^*$.

Lemma 2.23 *Suppose A is a square matrix. Then A is invertible if and only if A^* is invertible, and we have $(A^*)^{-1} = (A^{-1})^*$. Equivalently, the columns of A are linearly independent if and only if the rows of A are linearly independent.*

Proof: If A is invertible, then $Id = Id^* = (AA^{-1})^* = (A^{-1})^*A^*$, so A^* is invertible and $(A^*)^{-1} = (A^{-1})^*$. The same argument applied to A^* proves the converse because $(A^*)^* = A$.

The two statements are equivalent since the rows of A are the columns of of $\overline{A^*}$, and $\overline{A^*}$ is invertible if and only if A^* is invertible. □

Corollary 2.24 *If $T \stackrel{\mathbf{P},\mathbf{Q}}{\longleftrightarrow} A$, then T is invertible if and only if A is invertible, with $T^{-1} \stackrel{\mathbf{Q},\mathbf{P}}{\longleftrightarrow} A^{-1}$.* □

Determinant

There is a simple formula for the inverse of a 2×2 matrix:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \Rightarrow \quad A^{-1} = \begin{pmatrix} \frac{d}{ad-bc} & \frac{-b}{ad-bc} \\ \frac{-c}{ad-bc} & \frac{a}{ad-bc} \end{pmatrix} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}, \tag{2.52}$$

whenever $ad \neq bc$. The quantity $\det(A) \stackrel{\text{def}}{=} ad - bc$ is called the *determinant* of A, and is nonzero if and only if the two columns of A are linearly independent vectors.

More generally, for each $N > 0$, there is a determinant function $\det = \det(A)$ defined on matrices $A \in \mathbf{Mat}(N \times N)$ satisfying $\det(A) \neq 0$ if and only if A is invertible. Specifying enough extra conditions to construct it recursively ensures its uniqueness. One of these conditions is that $\det(AB) = \det(A)\det(B)$ for any matrices $A, B \in \mathbf{Mat}(N \times N)$. From these two properties, for example, we conclude

that $\det(Id) = 1$ and $\det(A^{-1}) = 1/\det(A)$. We will also require that $\det(A^*) = \det(A)$.

A square matrix $U \in \mathbf{Mat}(N \times N)$ is called *upper triangular* if $U(i,j) = 0$ whenever $i > j$. Such U is invertible if and only if there are no zeroes on the main diagonal, so we may define $\det(U) = U(1,1)U(2,2)\cdots U(N,N)$. It is left as an exercise to show that if U_1, U_2 are upper triangular, then their product is upper triangular with diagonal elements $U_1(n,n)U_2(n,n)$, $n = 1,\ldots,N$, and thus $\det()$ is multiplicative. Id is upper triangular, and this formula gives $\det(Id) = 1$, as required.

We likewise call a square matrix $L \in \mathbf{Mat}(N \times N)$ *lower triangular* if $L(i,j) = 0$ whenever $i < j$, and extend the definition of $\det()$ to this case by putting $\det(L) = L(1,1)L(2,2)\cdots L(N,N)$.

Rotations and rigid motions

Suppose $\mathbf{X}$ is an inner product space and $T : \mathbf{X} \to \mathbf{X}$ is an invertible linear transformation satisfying $T^*T = Id$ will be length preserving. This important subclass of linear transformations, for which $T^* = T^{-1}$, is called the *unitary transformations.*

Imagine for a moment how any linear transformation T acts on points in $\mathbf{X}$. $T\mathbf{0} = \mathbf{0}$ is a fixed point, so T is *origin-preserving.* Also, any subset $\mathbf{K} \subset \mathbf{X}$ gets transformed into the subset $T\mathbf{K} \stackrel{\text{def}}{=} \{T\mathbf{x} : \mathbf{x} \in \mathbf{K}\}$. For example, T maps the line segment $\mathbf{K} = \{\theta\mathbf{x} + (1-\theta)\mathbf{y} : 0 \le \theta \le 1\}$ between $\mathbf{x}$ and $\mathbf{y}$ to the set $\{\theta T\mathbf{x} + (1-\theta)T\mathbf{y} : 0 \le \theta \le 1\}$, which is the line segment between $T\mathbf{x}$ and $T\mathbf{y}$. Thus T is *line-preserving*; it sends lines to lines.

Now suppose that T is unitary. Since $\|T\mathbf{x}\|^2 = \langle T\mathbf{x}, T\mathbf{x}\rangle = \langle \mathbf{x}, T^*T\mathbf{x}\rangle = \langle x, x\rangle = \|x\|^2$, we have $\|T\mathbf{x}\| = \|\mathbf{x}\|$ for all $\mathbf{x} \in \mathbf{X}$. Then T is *length-preserving.* Also, for any two vectors $\mathbf{u}, \mathbf{v} \in \mathbf{X}$, the inner product $\langle T\mathbf{u}, T\mathbf{v}\rangle = \langle \mathbf{u}, T^*T\mathbf{v}\rangle = \langle \mathbf{u}, \mathbf{v}\rangle$ is preserved by T. Specifically, if $\mathbf{u}$ is perpendicular to $\mathbf{v}$, then $T\mathbf{u}$ is perpendicular to $T\mathbf{v}$. More generally, since $\langle \mathbf{u}, \mathbf{v}\rangle$, $\|\mathbf{u}\|$ and $\|\mathbf{v}\|$ determine the angle between $\mathbf{u}$ and $\mathbf{v}$, it follows that unitary transformations are *angle-preserving* as well as length-preserving. Therefore, applying a unitary transformation T produces a *rigid motion* which fixes the origin, and can be called a *rotation* about the origin[2].

The inverse of a unitary transformation is also unitary: using $T^{-1} = T^*$, we compute $(T^{-1})^* = (T^*)^* = T = (T^{-1})^{-1}$. In particular, the inverse of a rotation is also a rotation, but in the reverse direction.

Composing two rotations yields another rotation. In general, for any unitary transformations S, T, if $S^* = S^{-1}$ and $T^* = T^{-1}$, then $(ST)^* = T^*S^* = T^{-1}S^{-1} = (ST)^{-1}$, so ST is unitary.

If $T : \mathbf{X} \to \mathbf{X}$ is a unitary transformation on an N-dimensional inner product space $\mathbf{X}$ with orthonormal basis $\mathbf{P}$, then $T \stackrel{\mathbf{P},\mathbf{P}}{\longleftrightarrow} A$ where $A \in \mathbf{Mat}(N \times N)$ is a square matrix satisfying $A^* = A^{-1}$. We will call such matrices *unitary matrices.*

[2]The only other kind of rigid motion is *translation*, where $\mathbf{x} \mapsto \mathbf{x} + \mathbf{c}$ for some fixed $\mathbf{c}$. However, translation by $\mathbf{c} \neq \mathbf{0}$ is not a linear transformation because it does not fix $\mathbf{0}$.

Conversely, any unitary matrix gives a unitary transformation, once an orthonormal basis $\mathbf{P}$ is chosen.

For example, any two elementary basis vectors $\mathbf{e}_i, \mathbf{e}_j \in \mathbf{R}^N$, $i \neq j$, determine a plane, and the *Givens rotation* through an angle θ in that plane is given by the unitary matrix

$$G_{ij}(\theta) \stackrel{\text{def}}{=} I - (\mathbf{e}_{ii} + \mathbf{e}_{jj}) + \cos\theta\,(\mathbf{e}_{ii} + \mathbf{e}_{jj}) + \sin\theta\,(\mathbf{e}_{ij} - \mathbf{e}_{ji})\,, \tag{2.53}$$

which is the identity matrix modified at the four intersections of rows i, j and columns i, j to resemble the 2×2 plane rotation

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}.$$

Straightforward computation shows that $G_{ij}(\theta)^{-1} = G_{ij}(-\theta) = G_{ji}(\theta) = G_{ij}^*(\theta)$, proving that Givens rotations are unitary.

The determinant of a unitary matrix is a complex number if the scalars are complex, but it will have absolute value 1 since

$$|\det(A)|^2 = \overline{\det(A)}\det(A) = \det(A^*)\det(A) = \det(A^*A) = \det(Id) = 1.$$

2.2.4 Solving linear systems of equations

To find the multiplicative inverse of a matrix $A \in \mathbf{Mat}(N \times N)$, it is enough to find N vectors $\mathbf{x}_n$, $n = 1, \ldots, N$ such that $A\mathbf{x}_n = \mathbf{e}_n$, where $\mathbf{e}_n$ is an elementary basis vector. Then the matrix $B \in \mathbf{Mat}(N \times N)$ whose columns are $\mathbf{x}_1, \ldots, \mathbf{x}_N$ satisfies $AB = Id$, so $BA = Id$ and B is the inverse of A.

One standard method of solving the $N \times N$ system of linear equations $A\mathbf{x} = \mathbf{b}$, for an invertible matrix A, is to reduce A to upper triangular form A':

$$A = \begin{pmatrix} A(1,1) & A(1,2) & \ldots & A(1,N) \\ A(2,1) & A(2,2) & \ldots & A(2,N) \\ \vdots & \vdots & & \vdots \\ A(N,1) & A(N,2) & \ldots & A(N,N) \end{pmatrix}$$

$$\mapsto \quad A' = \begin{pmatrix} A'(1,1) & A'(1,2) & \ldots & A'(1,N) \\ 0 & A'(2,2) & \ldots & A'(2,N) \\ \vdots & & \ddots & \vdots \\ 0 & \ldots & 0 & A'(N,N) \end{pmatrix}. \tag{2.54}$$

The same operations applied to the vector $\mathbf{b}$ produce $\mathbf{b}'$, and $A\mathbf{x} = \mathbf{b}$ if and only if $A'\mathbf{x} = \mathbf{b}'$. But the primed system is easily solved by back substitution, starting with $x(N) = b'(N)/A'(N,N)$, as long as all the diagonal elements of A', namely $\{A'(n,n) : n = 1, \ldots, N\}$, are nonzero.

The reduction procedure is called *Gaussian elimination*, and it was introduced as Equation 2.9 in the proof of Theorem 2.2. In step one of this $(N-1)$-stage

process, we find a row index $\pi(1)$ such that $A(\pi(1),1) \neq 0$. This can always be done, since the N rows of A are linearly independent. If $|A(\pi(1),1)|$ is the largest absolute value of all in column 1, then finding $\pi(1)$ is called *partial pivoting*[3]. We then replace $A(i,j)$ by $A(i,j) - A(\pi(1),j)A(i,1)/A(\pi(1),1)$ for all $i = 2,\ldots,N$ except $i = \pi(1)$, and all $j = 2,\ldots,N$. Thus $A(i,1) = 0$ for all $i \neq \pi(1)$. Row $\pi(1)$ is then interchanged with row 1 to put the sole nonzero at the top of column 1. The coefficients are then renamed as the new matrix A_1, completing stage 1:

$$\begin{pmatrix} A(1,1) & A(1,2) & \cdots & A(1,N) \\ A(2,1) & A(2,2) & \cdots & A(2,N) \\ \vdots & & \ddots & \vdots \\ A(N,1) & A(N,2) & \cdots & A(N,N) \end{pmatrix} \mapsto \begin{pmatrix} A_1(1,1) & A_1(1,2) & \cdots & A_1(1,N) \\ 0 & A_1(2,2) & \cdots & A_1(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & A_1(2,2) & \cdots & A_1(N,N) \end{pmatrix}.$$

To preserve the linear system, we do the same to $\mathbf{b}$: replace $b(i)$ by $b(i) - b(\pi(1))A(i,1)/A(\pi(1),1)$ for all $i \neq \pi(1)$, and then interchange $b(1)$ with $b(\pi(1))$. Denote the result by $\mathbf{b}_1$; then $A\mathbf{x} = \mathbf{b} \iff A_1\mathbf{x} = \mathbf{b}_1$.

The same procedure is then applied to the $(N-1)\times(N-1)$ submatrix $\{A_1(i,j) : 2 \leq i,j \leq N\}$, determining $\pi(2)$ and $\{A_2(i,j) : 3 \leq i,j \leq N\}$. After a total of $N-1$ stages, the final matrix $A' \stackrel{\text{def}}{=} A_{N-1}$ is upper-triangular, and the vector $\mathbf{b}' \stackrel{\text{def}}{=} = \mathbf{b}_{N-1}$ is prepared.

Finally, the solution $\mathbf{x}$ to $A'\mathbf{x} = \mathbf{b}'$ is found by back substitution. Combining all steps gives the following algorithm:

Gaussian Elimination with Partial Pivoting

```
gepp( b[], A[], N ):
[ 1] For k=1 to N-1, do [2] to [10]
[ 2]    Find the largest value |A[m,k]|, for m in k,...,N
[ 3]    If A[m,k]==0, then print "A is singular" and stop.
[ 4]    If m!=k, then do [5] to [6]
[ 5]       For j=k to N, interchange A[m,j] and A[k,j]
[ 6]       Interchange b[m] and b[k]
[ 7]    For i=k+1 to N, do [8] to [10]
[ 8]       Replace A[i,k] /=  A[k,k]
[ 9]       For j=k+1 to N, replace A[i,j] -= A[i,k]*A[k,j]
[10]       Replace b[i] -= A[i,k]*b[k]
[11] For k=N down to 1, do [12] to [13]
[12]    For j=k+1 to N, replace b[k] -= A[k,j]*b[j]
[13]    Replace b[k] /= A[k,k]
```

Forward elimination with partial pivoting takes place in steps 1–10, while steps 11–13 are the back substitution. At the end, the coordinates of the solution vector

[3]Full pivoting seeks to put the largest absolute value of all at position $(1,1)$, by finding a column index $\nu(1)$ with $|A(\pi(1),\nu(1))|$ maximal among $|A(i,j)|$. With full pivoting, the solution vector comes out in the order $x(\nu(1)),\ldots,x(\nu(N))$.

x are left in `b[1],...,b[N]`. They do not need to be re-ordered since no column interchanges were performed on A.

The input matrix `A[]` is modified in stages until it is upper-triangular[4] and must at the end have nonzero elements on its diagonal. If this is cannot be done, the program will stop at step 3 with an error message claiming that A is singular, which means that for some **b**, $A\mathbf{x} = \mathbf{b}$ cannot be solved for **x**.

Inverses and LU decompositions

GEPP can be extended to find $A^{-1} \in \mathbf{Mat}(N \times N)$, whenever A is nonsingular. In the following implementation, lines with changes from the GEPP algorithm are marked with primes:

Matrix Inversion Using Partial Pivoting

```
mipp( B[], A[], N ):
[ 0'] Initialize B[i,j]=0, and for i=1 to N, let B[i,i] = 1
[ 1 ] For k=1 to N-1, do [2] to [10']
[ 2 ]    Find the largest value |A[m,k]|, for m in k,...,N
[ 3 ]    If A[m,k]==0, then print "A is singular" and stop.
[ 4 ]    If m!=k, then do [5] to [6']
[ 5 ]       For j=k to N, interchange A[m,j] and A[k,j]
[ 6']       For j=k to N, interchange B[m,j] and B[k,j]
[ 7 ]    For i=k+1 to N, do [8] to [10']
[ 8 ]       Replace A[i,k] /=  A[k,k]
[ 9 ]       For j=k+1 to N, replace A[i,j] -= A[i,k]*A[k,j]
[10']       For j=1 to N, replace B[i,j] -= A[i,k]*B[k,j]
[11 ] For k=N down to 1, do [12'] to [14']
[12']    For n=1 to N, do [13'] to [14']
[13']       For j=k+1 to N, replace B[k,n] -= A[k,j]*B[j,n]
[14']       Replace B[k,n] /= A[k,k]
```

The inverse of A will then be found in the double array `B[]`.

Actually, most of the information needed to reduce **b** to **b**$'$ during forward elimination is written into the lower-triangular part of `A[]`, rather than zeroes. If we also keep track of the interchanges $\pi(1), \pi(2)\ldots$, and apply them across entire rows of the matrix, then the operations on A may be performed in advance, preparing it for separate solution of $A\mathbf{x} = \mathbf{b}$ with **b** to be named later.

The result is called the *LU decomposition* of A. It is another encoding of A's inverse, somewhat cheaper to compute than A^{-1}, and with other advantages. In the implementation below, differences from the forward elimination part of GEPP are marked with double primes:

[4] At step 8, for efficiency, the subdiagonal element `A[i,k]` gets the precomputed multiplier `A[i,k]/A[k,k]`, needed many times thereafter. It should then be set to zero, if we insist that the returned array `A` actually be upper-triangular.

LU Decomposition with Partial Pivoting

```
lupp( A[], pi[], N ):
[0"] For k=1 to N, initialize pi[k] = k
[1 ] For k=1 to N-1, do [2] to [9]
[2 ]    Find the largest value |A[m,k]|, for m in k,...,N
[3 ]    If A[m,k]==0, then print "A is singular" and stop.
[4 ]    If m!=k, then do [5"] to [6"]
[5"]       For j=1 to N, interchange A[m,j] and A[k,j]
[6"]       Interchange pi[m] and pi[k]
[7 ]    For i=k+1 to N, do [8] to [9]
[8 ]       Replace A[i,k] /= A[k,k]
[9 ]       For j=k+1 to N, replace A[i,j] -= A[i,k]*A[k,j]
```

The *pivot* array `pi[1],..,pi[N]`, which defines $i \mapsto \pi(i)$, stores the interchange information. To use it, we re-index an input array `b` into the order `b[pi[1]]`, ..., `b[pi[N]]`, matching the partial pivoting rearrangement of rows of A. This is equivalent to multiplying $\mathbf{b}$ by the matrix $P = \sum_{i=1}^{N} \mathbf{e}_{i,\pi(i)}$, expressed here as a sum of elementary matrices. In other words, coefficient $P(i,j)$ is one if $j = \pi(i)$, but zero otherwise. If we apply P to a vector $\mathbf{b}$, we get a new vector $P\mathbf{b}$ whose i^{th} coordinate satisfies $P\mathbf{b}(i) = \mathbf{b}(\pi(i))$. If no interchanges are needed, we will have $\pi(i) = i$ for all $i = 1, 2, \ldots, N$, which is the *identity permutation.*

The array `A` returned by LUPP, which we shall denote by A'' to distinguish it from the original matrix A, contains an upper-triangular part U. Let us identify it as follows:

$$A'' = \begin{pmatrix} u(1,1) & u(1,2) & \cdots & u(1,N) \\ l(2,1) & u(2,2) & \cdots & u(2,N) \\ \vdots & & \ddots & \vdots \\ l(N,1) & \cdots & l(N,N-1) & u(N,N) \end{pmatrix};$$

$$U \stackrel{\text{def}}{=} \begin{pmatrix} u(1,1) & u(1,2) & \cdots & u(1,N) \\ 0 & u(2,2) & \cdots & u(2,N) \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & u(N,N) \end{pmatrix}. \tag{2.55}$$

But we obtained U from the row-interchanged matrix PA by applying a sequence of $N-1$ forward elimination steps: $L_{N-1} \cdots L_2 L_1 PA = U$, where the factors $L_1, \ldots$ can be written as matrices, using coefficients read directly from A'':

$$L_1 = \begin{pmatrix} 1 & & & \\ -l(2,1) & 1 & & \\ -l(3,1) & & \ddots & \\ \vdots & & & 1 \\ -l(N,1) & & & 1 \end{pmatrix}, \ldots,$$

$$L_{N-1} = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & -l(N,N-1) & 1 \end{pmatrix}. \tag{2.56}$$

Every coefficient not written out explicitly, or at least indicated by an ellipsis (dots), is zero.

Now note that L_k^{-1} is just L_k with the off-diagonal terms changed in sign:

$$L_k^{-1} = \begin{pmatrix} \ddots & & \\ & 1 & \\ & -l(k+1,k) & \\ & \vdots & \ddots \\ & -l(N,k) & \end{pmatrix}^{-1} = \begin{pmatrix} \ddots & & \\ & 1 & \\ & l(k+1,k) & \\ & \vdots & \ddots \\ & l(N,k) & \end{pmatrix}. \tag{2.57}$$

It is left as an exercise to prove that these formulas hold for all $k = 1, \ldots, N-1$. If we now define the product

$$L \stackrel{\text{def}}{=} L_1^{-1} L_2^{-1} \cdots L_{N-1}^{-1}, \tag{2.58}$$

then we may write $PA = LU$, with all the information needed to determine P, L, U returned in the arrays `pi` and `A`. It is left as an exercise to prove that L is in fact lower-triangular.

To solve $A\mathbf{x} = \mathbf{b}$ with this decomposition, we apply the interchanges, forward elimination, and back substitution steps only to b:

Linear System Solution from LU and Pivoting Data

```
sslu( b[], A[], pi[], N ):
[10] Permute b[1],...,b[N] to get b[pi[1]],...,b[pi[N]]
[11] For k=1 to N-1, do [12]
[12]    For i=k+1 to N, replace b[i] -= A[i,k]*b[k]
[13] For k=N down to 1, do [14] to [15]
[14]    For j=k+1 to N, replace b[k] -= A[k,j]*b[j]
[15]    Replace b[k] /= A[k,k]
```

This is equivalent to solving $PA\mathbf{x} = LU\mathbf{x} = P\mathbf{b}$ in three stages:

1. Apply all interchanges to $\mathbf{b}$ at once: $\mathbf{b} \mapsto P\mathbf{b} \stackrel{\text{def}}{=} \mathbf{b}_P$.
2. Apply forward elimination: $\mathbf{b}_P \mapsto L_1\mathbf{b}_P \mapsto \cdots \mapsto L_{N-1}\cdots L_1\mathbf{b}_P \stackrel{\text{def}}{=} \mathbf{b}_L$.
3. Solve the upper-triangular linear system $U\mathbf{x} = \mathbf{b}_L$ by back substitution.

After factoring $A = LU$, we may use Equation 2.58 and the multiplicative property of determinants to compute:

$$\det(A) = \det(L_1)^{-1} \cdots \det(L_{N-1})^{-1} \det(U) = \det(U),$$

since $\det(L_i) = 1$ for every $i = 1, \ldots, N-1$. But U is upper triangular, so

$$\det(A) = \det(U) = U(1,1) \cdots U(N,N).$$

Pivoting information is not needed; the determinant is just the product of the main diagonal elements left in the array `A[]` after LUPP:

Compute the Determinant of a Square Matrix

```
det( A[], N ):
[10]  Compute lupp(A[],pi[],N)
[11]  Let D = 1
[12]  For n=1 to N, replace D *= A[n,n]
[13]  Return D
```

This may also be done by appending lines `[11]-[13]` to `lupp()`.

2.2.5 Sparse matrices

Suppose that $A \in \mathbf{Mat}(M \times N)$ is the matrix representing some linear transformation with respect to fixed bases. To specify the transformation may require listing all $M \times N$ coefficients, if A is a so-called *full matrix*. To compute $\mathbf{y} = A\mathbf{x}$ in this case requires finding $y(m) = \sum_{n=1}^{N} A(m,n)x(n)$ for $m = 1, \ldots, M$, a total of $O(MN)$ operations of multiplication and addition. However, any zero matrix coefficient contributes nothing, and some useful matrices that are mostly zeroes, or *sparse*, can be specified with simple formulas.

In the examples below, we will denote by N either the index set $\{1, 2, \ldots, N\}$ or the set $\{0, 1, \ldots, N-1\}$. This is to accommodate the two different programming language conventions without duplicating formulas. Thus $\sum_{n \in N}$ is implemented either as $\sum_{n=1}^{N}$ or $\sum_{n=0}^{N-1}$, depending on the language. A similar interpretation will apply to the set M.

Diagonal Matrix. Suppose $N = M$. If $A(m,n) = 0$ unless $n = m$, then only the diagonal elements $A(n,n)$ for $n \in N$ need to be stored:

$$A = \begin{pmatrix} \ddots & & 0 \\ & A(n,n) & \\ 0 & & \ddots \end{pmatrix} = \sum_{n \in N} A(n,n)\mathbf{e}_{nn}. \tag{2.59}$$

Computing $A\mathbf{x}(m) = A(m,m)x(m)$ for $m \in N$ costs only $O(N)$ operations. The product of two diagonal matrices $\{A(n,n) : n \in N\}, \{B(n,n) : n \in N\}$ is another

diagonal matrix $AB = \{A(n,n)B(n,n) : n \in N\}$. A diagonal matrix A is invertible if and only if $A(n,n) \neq 0$ for all $n \in N$.

Permutation Matrix. Suppose $N = M$ and $A(m,n) \in \{0,1\}$[5] for all $n, m \in M$.

A is called a *permutation matrix* if A has a single 1 in each row and a single 1 in each column. In that case there will be a one-to-one and onto function $\pi : M \to M$ which gives the column index $n = \pi(m)$ of the 1 in row m. We may write $T \longleftrightarrow A \longleftrightarrow \pi$. Thus $T\mathbf{x}(m) = x(\pi(m))$ for $m \in M$ costs only $O(M)$ operations. The associated matrix is $A = \sum_{m \in M} \mathbf{e}_{m,\pi(m)}$. For example, the 3×3 matrix associated to $\pi(1) = 2, \pi(2) = 3, \pi(3) = 1$, which we may write $(1,2,3) \mapsto (2,3,1)$, is

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}. \tag{2.60}$$

The product of two permutation matrices $A \longleftrightarrow \pi_A$ and $B \longleftrightarrow \pi_B$ is another permutation matrix $AB \longleftrightarrow \pi_B \circ \pi_A$. Note that the order is reversed. The inverse of a permutation matrix is its transpose, which is also its adjoint since all coefficients are real, so a permutation matrix is a unitary linear transformation. This transpose is also a permutation matrix.

Tridiagonal Matrix. Suppose $N = M$. If $A(m,n) = 0$ unless $-1 \leq n - m \leq 1$,

then only the three diagonals in the middle of the matrix need to be stored. This takes $O(3N)$ memory locations. We compute $A\mathbf{x}(m) = A(m,m-1)x(m-1) + A(m,m)x(m) + A(m,m+1)x(m+1)$ for $m \in N$, handling the boundary terms with the convention $x(n) = 0$ if $n \notin N$ and $A(m,n) = 0$ if $m \notin N$ or $n \notin N$. This also costs only $O(3N)$ operations. The middle of the associated matrix is

$$A = \begin{pmatrix} & \ddots & & \ddots & & \ddots & & \\ \dots & 0 & A(m,m-1) & A(m,m) & A(m,m+1) & 0 & \dots \\ & & \ddots & & \ddots & & \end{pmatrix}. \tag{2.61}$$

We may also write $A = \sum_{n \in N}^{N} A(n,n)\mathbf{e}_{nn} + A(n+1,n)\mathbf{e}_{n+1,n} + A(n,n+1)\mathbf{e}_{n,n+1}$. Neither the product of tridiagonal matrices nor the inverse of a tridiagonal matrix is necessarily tridiagonal.

Banded Matrix. Suppose $N = M$. If $A(m,n) = 0$ unless $-p \leq n - m \leq q$,

then only the band of diagonals from p below to q above the main diagonal needs to be stored. This takes $O(N[1+p+q])$ memory locations. Computing $A\mathbf{x}(m) = \sum_{n=m-p}^{m+q} A(m,n)x(n)$ for $1 \leq m \leq N$ also costs only $O(N[1+p+q])$ operations.

[5] Such an A is called a $(0,1)$-*matrix*. Calculations with it require no multiplication operations, which is useful if multiplication circuitry is expensive or unavailable. A $(0,1,-1)$ *matrix* has a similar advantage.

The associated matrix in the indexing starting at 1 is

$$\begin{pmatrix} A(1,1) & A(1,2) & \dots & A(1,1+q) & 0 & \dots \\ A(2,1) & A(2,2) & A(2,3) & \dots & A(2,2+q) & 0 \\ \vdots & & & & & \\ A(1+p,1) & & & \ddots & & \ddots \\ 0 & \ddots & & & & \\ \vdots & & \ddots & & \ddots & \end{pmatrix}. \tag{2.62}$$

The only nonzero elements of a generic middle row m of this matrix are

$$A(m,m-p),\dots,A(m,m),\dots,A(m,m+q).$$

We can write this in either indexing convention as $A = \sum_{m\in N} \sum_{n=m-p}^{m+q} A(m,n)\mathbf{e}_{mn}$, if we agree to skip coefficients with out-of-range indices. The product of two banded matrices can have more bands than either factor, and the inverse of a banded matrix need not be banded.

Decimated Averages. Suppose $N = 2M$ is an even number. Indexing from 0, let $H \in \mathbf{Mat}(M \times 2M)$ be defined by $H(m,n) = 0$ unless $n = 2m$ or $n = 2m+1$, with $H(m,2m) = H(m,2m+1) = \frac{1}{2}$. Then $H\mathbf{u}(m) = \frac{1}{2}[u(2m) + u(2m+1)]$ for $0 \le m < M$; this is the sequence of averages of adjacent pairs in u. Then

$$H = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & \dots & 0 \\ \vdots & & & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix} = \frac{1}{2}\sum_{m\in M} (\mathbf{e}_{m,2m} + \mathbf{e}_{m,2m+1}). \tag{2.63}$$

Notice that $HH^* = \frac{1}{2}Id \in \mathbf{Mat}(M \times M)$, while $2H^*H \in \mathbf{Mat}(2M \times 2M)$ is an orthogonal projection from $\mathbf{E}^{2M}$ onto the M-dimensional subspace spanned by the rows of H.

Decimated Differences. Suppose $N = 2M$ is an even number. Indexing from 0, let $G \in \mathbf{Mat}(M \times 2M)$ be defined by $G(m,n) = 0$ unless $n = 2m$ or $n = 2m+1$, with $G(m,2m) = -1$ and $G(m,2m+1) = +1$. Then $G\mathbf{u}(m) = u(2m+1) - u(2m)$ for $0 \le m < M$; this is the sequence of differences of adjacent pairs in u. Then

$$G = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & -1 & 1 & 0 & \dots & 0 \\ \vdots & & & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & 0 & -1 & 1 \end{pmatrix} = \sum_{m\in M} (\mathbf{e}_{m,2m+1} - \mathbf{e}_{m,2m}). \tag{2.64}$$

Notice that $GG^* = 2Id \in \mathbf{Mat}(M \times M)$, while $\frac{1}{2}G^*G \in \mathbf{Mat}(N \times N)$ is an orthogonal projection from $\mathbf{E}^{2M}$ onto the M-dimensional subspace spanned by the rows of G. With the decimated averages matrix H, we have two properties:

Independence: $GH^* = HG^* = 0 \in \mathbf{Mat}(M \times M)$. Thus the column space of G^* is orthogonal to the column space of H^*. As a consequence, for all $\mathbf{x} \in \mathbf{E}^{2M}$, we have $G^*G\mathbf{x} \perp H^*H\mathbf{x}$.

Completeness: $2H^*H + \frac{1}{2}G^*G = Id \in \mathbf{Mat}(2M \times 2M)$. Thus every $\mathbf{x} \in \mathbf{E}^{2M}$ may be written as $\mathbf{x} = \mathbf{s} + \mathbf{d}$, for some $\mathbf{s}$ in the M-dimensional column space of H^* and some $\mathbf{d}$ in the M-dimensional column space of G^*.

Running Averages. Fix $K > 1$ and for $0 \leq m \leq N - K$ and $0 \leq n < N$ let

$$A(m,n) = \begin{cases} \frac{1}{K}, & \text{if } m \leq n < m + K; \\ 0, & \text{otherwise.} \end{cases} \tag{2.65}$$

Then $A\mathbf{x}(m)$ is the average of the K components of $\mathbf{x}$ starting with $x(m)$. The associated matrix is

$$\begin{pmatrix} \frac{1}{K} & \frac{1}{K} & \dots & \frac{1}{K} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{K} & \dots & \frac{1}{K} & \frac{1}{K} & 0 & \dots & 0 \\ \vdots & & & & \ddots & & & \vdots \\ 0 & 0 & \dots & 0 & \frac{1}{K} & \frac{1}{K} & \dots & \frac{1}{K} \end{pmatrix} = \frac{1}{K} \sum_{m \in M} \sum_{n=m}^{m+K-1} \mathbf{e}_{m,n}. \tag{2.66}$$

As usual, out-of-range coefficients are set to zero.

2.3 Exercises

1. How many vertices and edges are there in the 5-cube, the unit cube in Euclidean 5-space? Find a formula in terms of N for the number of vertices and edges of an N-cube.

2. Find an example subspace of $\mathbf{R}^N$ of dimension k for each $k = 0, 1, \dots, N$.

3. Show that the system of inequalities 2.14, 2.15, and 2.16 is sharp for every N by finding example vectors in $\mathbf{C}^N$ that give equality.

4. Prove that the following are equivalent:

   (i) $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ and $\|a\mathbf{x}\| = |a| \, \|\mathbf{x}\|$, for all vectors $\mathbf{x}, \mathbf{y}$ and scalars a;

   (ii) $\|\mathbf{x} + \mathbf{y}\| \leq |a| \, \|\mathbf{x}\| + |b| \, \|\mathbf{y}\|$, for all vectors $\mathbf{x}, \mathbf{y}$ and scalars a, b.

5. Show that for any subset $\mathbf{Y}$ of an inner product space $\mathbf{X}$, we have $\mathbf{Y} \cap \mathbf{Y}^\perp = \{\mathbf{0}\}$, and $\mathbf{Y} \subset \left(\mathbf{Y}^\perp\right)^\perp$. (This is Lemma 2.5.)

6. Suppose that $\mathbf{Y} = \text{span}\,\{\mathbf{y}_n : n = 1, \dots, N\}$. Show that if $\langle \mathbf{x}, \mathbf{y}_n \rangle = 0$ for all n, then $\mathbf{x} \in \mathbf{Y}^\perp$. (This is Lemma 2.6.)

7. Find an orthonormal basis for the subspace of $\mathbf{E}^4$ spanned by the vectors $\mathbf{x} = (1,1,1,1)$, $\mathbf{y} = (1,0,1,0)$, and $\mathbf{z} = (1,0,0,1)$.

8. Let $\{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$ be the standard basis vectors of $\mathbf{E}^N$ (defined by Equation 2.7) and, for notational convenience, put $\mathbf{e}_{N+1} \stackrel{\text{def}}{=} 0$. Show that $\{\mathbf{f}'_n = \mathbf{e}_n - \mathbf{e}_{n+1} : n = 1, \ldots, N\}$ (see Equation 2.34) is the biorthogonal dual of the basis $\{\mathbf{f}_n = \mathbf{e}_1 + \cdots + \mathbf{e}_n : n = 1, \ldots, N\}$ (see Equation 2.8).

9. Prove that the inner product given by the Riemann integral

$$\langle u, v \rangle \stackrel{\text{def}}{=} \int_0^1 \overline{u(t)}\, v(t)\, dt$$

is defined for all continuous functions $u = u(t)$ and $v = v(t)$ on $0 \leq t \leq 1$, and is Hermitean symmetric, nondegenerate, and linear.

10. Prove that the operator norm $\|\cdot\|_{\text{op}}$ satisfies the norm axioms and is submultiplicative.

11. Prove that the Hilbert–Schmidt norm $\|\cdot\|_{\text{HS}}$ defined by Equation 2.44 satisfies the norm axioms and is submultiplicative.

12. Suppose that $A, B \in \mathbf{Mat}(M \times N)$. Prove Equation 2.47:

$$\langle A, B \rangle = \operatorname{tr}(A^*B) = \operatorname{tr}(BA^*) = \overline{\operatorname{tr}(AB^*)} = \overline{\operatorname{tr}(B^*A)}.$$

Deduce that $\|A\|_{\text{HS}} = \sqrt{\operatorname{tr}(A^*A)} = \sqrt{\operatorname{tr}(AA^*)}$.

13. Suppose that $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$ is a finite orthonormal subset of an inner product space $\mathbf{X}$. Prove that the transformation $P_{\mathbf{Y}}$ defined by

$$P_{\mathbf{Y}}\, \mathbf{x} \stackrel{\text{def}}{=} \sum_{k=1}^{N} \langle \mathbf{y}_k, \mathbf{x} \rangle\, \mathbf{y}_k$$

is an orthogonal projection onto $\operatorname{span} \mathbf{Y}$. (This establishes Equation 2.49.)

14. Suppose that $\mathbf{X}$ is an infinite-dimensional inner product space and $\mathbf{Y} = \{\mathbf{y}_n : n \in \mathbf{Z}^+\}$ is an orthonormal subset of $\mathbf{X}$. For any $\mathbf{x} \in \mathbf{X}$ and all $n = 1, 2 \ldots$, put $c(n) = \langle \mathbf{y}_n, \mathbf{x} \rangle$ and define the partial sums

$$\mathbf{x}_N = \sum_{n=1}^{N} c(n) \mathbf{y}_n, \qquad N = 1, 2, \ldots.$$

a. Prove *Bessel's inequality*: For any $N = 1, 2, \ldots$,

$$\sum_{n=1}^{N} |c(n)|^2 = \|\mathbf{x}_N\|^2 \leq \|\mathbf{x}\|^2.$$

b. Prove *Parseval's formula*:

$$\lim_{N\to\infty} \|\mathbf{x}_N\|^2 = \sum_{n=1}^{\infty} |c(n)|^2 = \|\mathbf{x}\|^2 \quad \Longleftrightarrow \quad \lim_{N\to\infty} \|\mathbf{x} - \mathbf{x}_N\| = 0.$$

15. Prove that the linear transformation $T : \ell^2 \to \ell^2$ defined by

$$T(x_1, x_2, \ldots) = (x_1, 2x_2, \ldots, nx_n, \ldots)$$

is unbounded. (This is example 5 in section 2.2.)

16. Prove that $G_{ij}(\theta)^{-1} = G_{ij}(-\theta) = G_{ji}(\theta) = G_{ij}^*(\theta)$, where $i \neq j$ and $G_{ij}(\theta)$ is a Givens rotation as defined in Equation 2.53.

17. Prove Equation 2.57, namely, that for all $k = 1, \ldots, N-1$,

$$\begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & l(k{+}1,k) & & \\ & & \vdots & \ddots & \\ & & l(N,k) & & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l(k{+}1,k) & & \\ & & \vdots & \ddots & \\ & & -l(N,k) & & 1 \end{pmatrix}$$

18. Prove that the product of two lower-triangular $N \times N$ matrices is lower-triangular, and that the product of two upper-triangular $N \times N$ matrices is upper-triangular.

19. Write a computer program to visualize the rotated four-dimensional unit cube. Have it perform the following steps:

    1. Choose six angles $\{\theta_{ij} : 1 \leq i < j \leq 4\}$ from user input.
    2. Apply the six Givens rotations $\{G_{ij}(\theta_{ij}) : 1 \leq i < j \leq 4\}$, in a fixed order, to all the vertices $\mathbf{Q}_0, \ldots, \mathbf{Q}_{15}$ of the four-dimensional unit cube, as listed in Table 2.2.
    3. Find the orthogonal projection into xyz-space of the rotated vertices.
    4. Find the perspective projection onto the xy-plane of the points in xyz-space.
    5. Connect the vertices in the xy-plane with the edges listed in Table 2.2, and display the resulting graphic.

2.4 Further Reading

- Wolfgang Boehm and Hartmut Prautzsch. *Geometric Concepts for Geometric Design.* A K Peters, Ltd., Natick, Massachusetts, 1994. ISBN 1-56881-004-0.
- David Holzgang. *Understanding PostScript Programming.* Sybex, San Francisco, 1987. ISBN 0-89588-396-1.
- Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg and Didier L. LeGall. *MPEG Video Compression Standard.* Chapman and Hall, New York, 1996. ISBN 0-412-08771-5.
- Gilbert Strang. *Linear Algebra and its Applications.* Harcourt, Brace, Jovanovich, San Diego, third edition, 1988. ISBN 0-15-551005-3.
- Daniel Zwilllinger, editor. *Standard Mathematical Tables and Formulae.* CRC Press, Boca Raton, Florida, 30th edition, 1996. ISBN 0-8493-2479-3.

Chapter 3

Time and Frequency

3.1 Fourier Analysis

The amount of work needed to compute a transformation of a function depends quite heavily on the way it is represented by the computer. There are many advantages to using combination of more basic functions. In the early 19th century, Jean-Baptiste Joseph Fourier chose sines and cosines as building blocks because he could obtain easy formulas for their derivatives. His work, augmented by many other researchers, showed that any given smooth function can be approximated arbitrarily well by a finite linear combination of sines and cosines. The number of components depends only on the smoothness of the target function and the desired degree of approximation. Such expansions provide compact descriptions of complicated functions and simplify the transmission and display of multimedia information.

In addition, a *Fourier sine component* such as $A\sin(2\pi\omega t - \delta)$ has three identifying parameters with physical meaning: its *amplitude* A, its *frequency* ω, and its *initial phase* δ. Thus, the analysis of a complicated function into its Fourier sine and cosine components splits it into parts labeled for content.

The formulas used in Fourier analysis are the same as those for finding expansions of vectors in an inner product space: let $\{e_n = e_n(t)\}$ be an orthonormal basis for an inner product space X of functions, and let $f = f(t)$ be an arbitrary function in X. Then

$$f = \sum_n \langle e_n, f \rangle \, e_n.$$

There are various technical difficulties to overcome when the index n ranges over an infinite set such as $\mathbf{Z}$, even though any given Fourier expansion will ultimately be truncated to a finite series for computation. Also, Fourier's choice of sines and cosines for the basis $\{e_n\}$ has certain peculiarities, and some seemingly harmless combinations with other approximations can result in large errors. Still, discrete Fourier transforms are the single most important process applied to digital signals.

3.1.1 Periodic functions

A function $f = f(t)$ defined at all real t is called *periodic*, or *T-periodic*, if there is some $T > 0$ such that

$$f(t) = f(t+T), \qquad \text{for every number } t \in \mathbf{R}. \tag{3.1}$$

If such a T exists, then $f(t) = f(t+2T) = f(t+3T) = f(t-T) = f(t-2T)$, and so on, for every $t \in \mathbf{R}$. Any such T is called a *period* of f. Some natural examples are trigonometric functions like sine and cosine, which are 2π-periodic. Here it is common to refer to "the" period since 2π is the least positive T that works. Another easy example is any constant function, but this is T-periodic for all $T > 0$ and has no least period $T > 0$.

All values of a T-periodic function are determined by its values on the interval $[0, T)$, or more generally on any *period interval* $[a, a+T)$ or $(a, a+T]$. Thus, computing $\sin t$ for $0 \le t < 2\pi$ or equivalently for $-\pi < t \le \pi$ determines its values at every t.

The *T-periodization* of an arbitrary function $f = f(t)$ is the new function defined by the infinite sum

$$f_T(t) = \sum_{k=-\infty}^{\infty} f(t+kT), \qquad \text{for any number } t \in \mathbf{R}. \tag{3.2}$$

If f is continuous and decreases to 0 rapidly enough as t tends to $\pm\infty$, then the series converges and the result is T-periodic:

Lemma 3.1 *Suppose that $f = f(t)$ is a function defined on $\mathbf{R}$ such that the integral $\int_a^b f(t)\,dt$ exists on every interval $[a, b] \subset \mathbf{R}$, and such that $|f(t)| < C/|t|^2$ for some constant $C > 0$ and all $t \in \mathbf{R}$. Then*

a. The improper integral $\int_{-\infty}^{\infty} f(t)\,dt$ exists.

b. For any $T > 0$, the T-periodization f_T of Equation 3.2 exists at every t, is T-periodic, is integrable on each period interval $[a, a+T]$, and satisfies

$$\int_{-\infty}^{\infty} f(t)\,dt = \int_0^T f_T(t)\,dt = \int_a^{a+T} f_T(t)\,dt, \qquad \text{for every } a \in \mathbf{R}.$$

Proof: Define $I_n = \int_{-n}^{n} f(t)\,dt$ for every $n \in \mathbf{Z}^+$. Then if $m \ge n \ge N \in \mathbf{Z}^+$, we have

$$\begin{aligned} |I_m - I_n| &= \left| \int_{-m}^{m} f(t)\,dt - \int_{-n}^{n} f(t)\,dt \right| \\ &\le \int_{-m}^{-n} |f(t)|\,dt + \int_{n}^{m} |f(t)|\,dt \\ &\le 2C \int_N^{\infty} \frac{dt}{t^2} = 2C/N, \end{aligned}$$

so $\{I_n : n = 1, 2, \ldots\}$ is a Cauchy sequence. Call its limit I_∞; then similar estimates show that for every $\epsilon > 0$ there is an $N_\epsilon \in \mathbf{Z}^+$ such that $|\int_{-a}^{b} f(t)\,dt - I_\infty| < \epsilon$ whenever $a, b > N_\epsilon$. Thus $\int_{-\infty}^{\infty} f(t)\,dt = I_\infty$ exists.

For $t \in [0, T]$, the series $f_T(t) \stackrel{\text{def}}{=} \sum_n f(t + nT)$ converges absolutely by comparison with $\sum_{n \neq 0} C/|nT|^2$. For other intervals $[nT, (n+1)T]$, the same argument works after a change of summation index. Hence f_T is defined on all of $\mathbf{R}$. The same argument also shows that f_T is T-periodic.

To compare integrals, note that for any two positive integers N, M, we have

$$\begin{aligned}\int_0^T \sum_{n=-N}^{M-1} f(t + nT)\,dt &= \sum_{n=-N}^{M-1} \int_0^T f(t + nT)\,dt \\ &= \sum_{n=-N}^{M-1} \int_{nT}^{(n+1)T} f(t)\,dt = \int_{-NT}^{MT} f(t)\,dt.\end{aligned}$$

Taking the limit $N, M \to \infty$ gives the first claimed equality.

Finally, write $a = a' + nT$, where $n = \lfloor a/T \rfloor$ is the greatest integer in a/T, and $a' = a - nT$ is the nonnegative fractional part left over. Then since $f_T(t + nT) = f_T(t)$ at every $t \in \mathbf{R}$, we have

$$\int_a^{a+T} f_T(t)\,dt = \int_{a'}^{a'+T} f_T(t)\,dt = \int_{a'}^{T} f_T(t)\,dt + \int_T^{a'+T} f_T(t)\,dt = \int_0^T f_T(t)\,dt.$$

The last step is the change of variable $t \leftarrow t + T$ followed by recombining the resulting integrals. □

More careful analysis reveals that the weaker condition of absolute integrability, which requires some decrease on average as $t \to \pm\infty$, is enough to give the same conclusion. See, for example, Theorem 10.26 on page 309 of Apostol's *Mathematical Analysis*, in the further readings.

For the first example, we consider three periodizations of the hat function h with $k = n = 0$ defined in Equation 2.22:

$$h(t) = \begin{cases} 0, & \text{if } t < -1 \text{ or } t > 1; \\ t + 1, & \text{if } -1 \leq t \leq 0; \\ 1 - t, & \text{if } 0 \leq t \leq 1. \end{cases}$$

This vanishes outside $[-1, 1]$, so it certainly satisfies the decay-at-infinity condition.

$T = 1$. The 1-periodization of h is

$$h_1(t) = \sum_{k \in \mathbf{Z}} h(t + k) = h(t + k_1) + h(t + k_2),$$

where k_1, k_2 are integers satisfying $t + k_1 \in [-1, 0)$ and $t + k_2 \in [0, 1)$. Thus $-1 - t \leq k_1 < -t$ and $-t \leq k_2 < -t + 1$, so $k_2 = \lfloor -t - 1 \rfloor$ and $k_2 = \lfloor -t \rfloor = k_1 + 1$. But then $h_1(t) = (t + k_1) + 1 + 1 - (t + k_2) = 1$ for all $t \in \mathbf{R}$: the 1-periodized hat function is the constant function 1.

$T = 2$. For each $t \in \mathbf{R}$, suppose that $t + 2k \in [-1, 1)$. Then $\frac{-1-t}{2} \leq k < \frac{1-t}{2} = \frac{-1-t}{2} + 1$, so there is a unique $k_0 = \lfloor \frac{-t-1}{2} \rfloor \in \mathbf{Z}$ such that $h_1(t) = h(t + 2k_0)$. The graph of h_2 consists of adjacent, nonoverlapping copies of h.

$T = 3$. Suppose that $t + 3k \in [-1, 1)$ for some integer k. Then $-3k - 1 \leq t < -3k+1$, so t must be within 1 of an integer multiple of 3. If not, then $h_3(t) = 0$ is given by a sum of zeroes. The graph of h_3 thus consists of nonoverlapping copies of h, centered at integer multiples of 3, separated by gaps of the zero function on intervals of the form $(3k + 1, 3k + 2)$, $k \in \mathbf{Z}$.

For another example, consider the function

$$f(t) = \begin{cases} 2^{-t}, & \text{if } t \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

This satisfies the rapid decay condition $|f(t)| < 1/t^2$ as $t \to \pm\infty$, and it is certainly integrable on any interval $[a, b]$. To compute the 1-periodization, note that $t + k \geq 0 \iff k \geq -t \iff k \geq \lceil -t \rceil = -\lfloor t \rfloor$, so that the substitution $k \leftarrow k' - \lfloor t \rfloor$ gives

$$\sum_{k \in \mathbf{Z}} f(t + k) = \sum_{k'=0}^{\infty} 2^{-k' + \lfloor t \rfloor - t} = 2 \cdot 2^{\lfloor t \rfloor - t},$$

which consists of adjacent, nonoverlapping copies of the portion of $g(t) = 2 \cdot 2^{-t}$ that lies over $[0, 1)$.

It is difficult to derive properties of a function from its periodization since every value of a periodized function is the sum of infinitely many values of the original function at arbitrarily distant points.

A function $f = f(x, y)$ of two variables may be S-periodized in the first and T-periodized in the second by the formula $f_{ST}(x, y) \stackrel{\text{def}}{=} \sum_{m,n \in \mathbf{Z}} f(x + mT, y + nS)$. In general, given a vector of periods $\mathbf{T} = (T(1), \ldots, T(N))$, we may periodize a function $f = f(\mathbf{x})$ of N variables $\mathbf{x} = (x(1), \ldots, x(N))$ by

$$f_{\mathbf{T}}(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{\mathbf{k} \in \mathbf{Z}^N} f(x(1) + k(1)T(1), \ldots, x(N) + k(N)T(N)), \tag{3.3}$$

where the sum runs over all lists $\mathbf{k} = (k(1), \ldots, k(N))$ of N integers.

3.1.2 Localization

A periodic function f is like the signal from a stuck record, repeating the same notes over and over. A more general signal will change with time and should be *localized* for study. The simplest way to localize is to restrict to an interval of interest, setting the function to zero outside. Thus, suppose $f = f(t)$ is defined at all real t. The *localization* of f to a closed and bounded interval $I = [a, b] \subset \mathbf{R}$ is the function

$$\mathbf{1}_I f(t) = \begin{cases} f(t), & \text{if } t \in I, \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

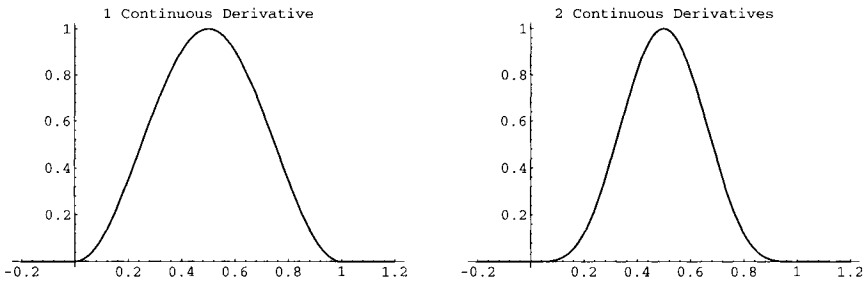


Figure 3.1: Cosine-based window functions with 1 and 2 continuous derivatives

In other words, localization to I multiplies f by $\mathbf{1}_I$, the indicator function of the interval. The definition may also be applied to other kinds of intervals, like $I = (a, b)$ or $I = [a, b)$.

In general, a function $f = f(t)$ is said to have *compact support* if there is some closed and bounded interval $I = [a, b]$ such that $f(t) = 0$ for all $t \notin I$. Such an f is said to be *supported on* I. A simple example is the zero function, which has compact support using any I. Less special is $\mathbf{1}$, which has compact support and is supported on $[0, 1]$. The function $g(t) = e^{-t^2}$, which is never zero and therefore not compactly supported, may be described as supported on all of $\mathbf{R}$. But the restriction of any function f to a bounded interval I will have compact support, and $\mathbf{1}_I f$ will be supported in I.

Even if f is continuous, its localization need not be: there may be jump discontinuities at the endpoints a and b of I. But if $f(a) = f(a+) = 0$ and $f(b) = f(b-) = 0$ and f is continuous at each $t \in (a, b)$, then $\mathbf{1}_I f$ is continuous. If in addition f satisfies a Lipschitz condition on I, then the localization $\mathbf{1}_I f$ will satisfy a Lipschitz condition on all of $\mathbf{R}$.

A more sophisticated solution is to smooth the cut by pinching the ends down to zero. Let $u = u(t)$ be a "window function" such as the one defined below, which is called a *Hanning window*:

$$u(t) = \begin{cases} 0, & \text{if } t < 0 \text{ or } t > 1; \\ \frac{1}{2} - \frac{1}{2}\cos 2\pi t, & \text{if } 0 \le t \le 1. \end{cases} \tag{3.5}$$

its graph is plotted in the left half of Figure 3.1. This function u has one continuous derivative: $u'(0+) = 0 + \pi B \sin(2\pi 0+) = 0 = u'(0-)$ and $u'(1-) = 0 + \pi B \sin(2\pi -) = 0 = u'(1+)$, so the derivatives of the pieces match up at $t = 0$ and $t = 1$.

The Hanning window is one member of a family of functions of the form

$$u(t) = \begin{cases} A - B\cos 2\pi t + C\cos 4\pi t, & \text{if } t \in [0, 1], \\ 0, & \text{otherwise.} \end{cases}$$

Such a u can be made smoother by choosing $C \neq 0$ as follows: note that for all windows in the family, $u(0+) = A - B + C = u(1-)$, $u'(0+) = u'(1-) = 0$, and

$$u''(0+) = 4\pi^2 B - 16\pi^2 C = u''(1-).$$

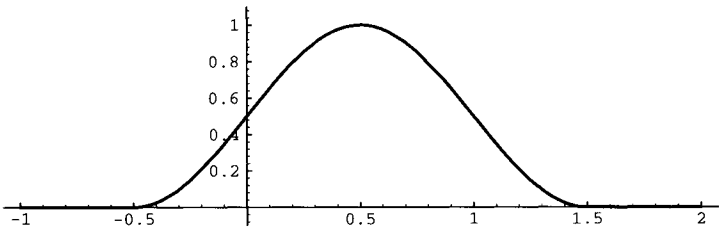


Figure 3.2: Wide Hanning window function.

Since $u(0-) = u'(0-) = u''(0-) = 0$ and $u(1+) = u'(1+) = u''(1+) = 0$, continuity at 0 and 1 of u and its first two derivatives requires $A - B + C = 0$ and $B - 4C = 0$. It is also convenient for u to have maximum value 1, but this must occur at the critical point $t = \frac{1}{2}$ where $u(t) = A + B + C$. The resulting system of three linear equations has a unique solution $A = 0.375$, $B = 0.500$, and $C = 0.125$, which gives the window illustrated in the right half of Figure 3.1. It has an additional nice property since $\cos 4\pi t = 2\cos^2 2\pi t - 1$:

$$u(t) = \frac{1}{8}(3 - 4\cos 2\pi t + \cos 4\pi t) = \frac{1}{4}(1 - \cos 2\pi t)^2 \geq 0,$$

for all $t \in [0, 1]$. Evidently, this window is the square of the Hanning window, suggesting a generalization: let $u^n(t) = [(1 - \cos 2\pi t)/2]^n$ for $t \in [0, 1]$, with $u^n(t) = 0$ elsewhere. Then u^n will be continuous and will have n continuous derivatives, as well as being nonnegative with maximum value 1 at $t = \frac{1}{2}$. We may expand u^n as a sum of cosines; it will have $n + 1$ terms since

$$\left[\frac{1 - \cos 2\pi t}{2}\right]^n = 2^{-n} \sum_{k=0}^{n} (-1)^k \binom{n}{k} \cos 2\pi k t.$$

This expansion is called the *Fourier series* for u^n.

Rather than pinch off the signal within the interval of interest to get a smooth function, we can mix in parts of the signal just outside the interval by using a wider window. For example, let $I = [0, 1]$ and define $w = w(t)$ by

$$w(t) = \begin{cases} 0, & \text{if } t < -\frac{1}{2} \text{ or } t > \frac{3}{2}; \\ (1 + \sin \pi t)/2, & \text{if } -\frac{1}{2} \leq t \leq \frac{3}{2}. \end{cases} \tag{3.6}$$

Its graph is plotted in Figure 3.2. This is just the Hanning window composed with the substitution $t \leftarrow \frac{1}{2}(t + \frac{1}{2})$, and has the same smoothness: continuity and one continuous derivative.

Periodic extension

A function localized to a bounded interval I can always be periodized since the sum in Equation 3.2 will be finite for any $T > 0$. A natural choice of period is $T = |I|$,

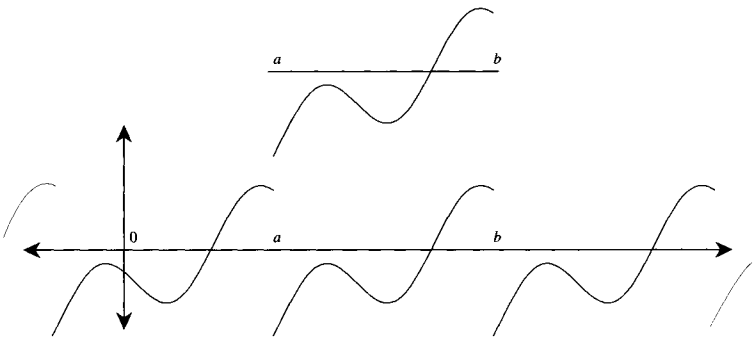


Figure 3.3: Top: Graph of an example function f defined on $I = [a, b)$. Bottom: Partial graph of the periodic extension f_I of f. Note the jump discontinuities and the agreement with f on I.

for which the sum will have exactly one[1] potentially nonzero term. So, define the *periodic extension of f from $I = [a, b)$* to be

$$f_I(t) = \sum_{k \in \mathbf{Z}} (\mathbf{1}_I f)(t + k|I|). \tag{3.7}$$

In fact $f_I(t) = f(t_0)$ for all $t \in \mathbf{R}$, where $t_0 = t_0(t) \in I$ is the unique number such that $t - t_0$ is an integer multiple of $|I| = b - a$. This relationship may be described by the equation

$$t_0 = t \bmod I. \tag{3.8}$$

Then f_I is $|I|$-periodic since for all $t \in \mathbf{R}$, $f_I(t + |I|) = f((t + |I|) \bmod I) = f(t \bmod I) = f_I(t)$. Also, f_I agrees with f on I. Periodic extension is depicted in Figure 3.3.

Even if f is continuous, f_I need not be unless there are no jump discontinuities at the points $t = a + k(b - a)$ "between periods," where k is an integer. But continuity is guaranteed simply by requiring $f(a) = f(b)$:

Lemma 3.2 *Suppose that $f = f(t)$ is defined on $[a, b]$ with $f(a) = f(b)$. Put $I = [a, b)$ and let f_I be the periodic extension of f from I. Then*

1. *If f is continuous on $[a, b]$, then f_I is continuous on $\mathbf{R}$.*
2. *If $f \in \mathbf{Lip}[a, b]$, then $f_I \in \mathbf{Lip}(\mathbf{R})$.*

Proof: For 1: Given $t \in \mathbf{R}$, let $t_0 = t \bmod I$. If $t_0 \in (a, b)$, then $f_I(t-) = f(t_0-) = f(t_0) = f_I(t)$, and $f_I(t+) = f(t_0+) = f(t_0) = f_I(t)$, since f is continuous at t_0. Thus f_I will be continuous at t.

[1]Endpoints may conspire, so we will always assume that one of them is missing: $I = [a, b)$ or $I = (a, b]$.

Otherwise $t_0 = a$, and then $f_I(t-) = f_I(b-) = f(b-) = f(b) = f(a) = f_I(a)$, and $f_I(a+) = f(a+) = f(a) = f_I(a)$, since f is continuous from the left at a and from the right at b. Again, f_I will be continuous at t.

For 2: Given $s, t \in \mathbf{R}$, let $s_0 = s \bmod I$ and $t_0 = t \bmod I$. If $|s_0 - t_0| \leq |s - t|$, then $|f_I(s) - f_I(t)| = |f(s_0) - f(t_0)| \leq C|s_0 - t_0| \leq C|s - t|$, where C is the constant in the Lipschitz condition for f.

Otherwise, $|s_0 - t_0| > |s - t|$. We may assume without loss of generality that $a \leq s_0 \leq t_0 < b$, and then $|s_0 - t_0 + (b - a)| < |s - t|$. Since $f(a) = f(b)$, we may write $|f_I(s) - f_I(t)| = |f(s_0) - f(t_0)| = |f(s_0) - f(a) + f(b) - f(t_0)| \leq |f(s_0) - f(a)| + |f(b) - f(t_0)| \leq C|s_0 - a| + C|b - t_0| = C|s_0 - t_0 + (b - a)| < C|s - t|$. □

To insure that f_I has as many derivatives as f, we must match the derivatives at a and b as well:

Corollary 3.3 *Suppose that $f = f(t)$ is defined on $[a, b]$ with $f(a) = f(b)$. Put $I = [a, b)$ and let f_I be the periodic extension of f from I. Suppose further that f is d-times differentiable on (a, b), with the endpoint limits agreeing for every $0 \leq n \leq d$ as follows:*

$$f^{(n)}(a+) \stackrel{\text{def}}{=} \lim_{t \to a+} f^{(n)}(t) = \lim_{t \to b-} f^{(n)}(t) \stackrel{\text{def}}{=} f^{(n)}(b-).$$

Then f_I is d-times differentiable on all of $\mathbf{R}$.

Proof: Apply Lemma 3.2 successively to $f, f', \ldots, f^{(d)}$, and define $f^{(n)}(a) = f^{(n)}(b) = f^{(n)}(a+) = f^{(n)}(b-)$ for $n = 0, 1, \ldots, d$. □

A function $f = f(t)$ is said to be d-times *continuously differentiable on* $\mathbf{R}$, or in the class $C^d(\mathbf{R})$, for some nonnegative integer d, if $f(t)$ and the derivatives $f'(t), f''(t), \ldots, f^{(d)}(t)$ exist at each $t \in \mathbf{R}$ and give functions $f, f', \ldots, f^{(d)}$ continuous on $\mathbf{R}$. Similar definitions can be made for the class $C^d(I)$ on any open interval $I = (a, b)$. Such functions are also said to be *smooth of degree d*.

If f belongs to $C^d(\mathbf{R})$ for all integers $d \geq 0$, we say that f is *smooth*, without qualification, or that f belongs to $C^\infty(\mathbf{R})$. Some examples of smooth functions are: polynomials, including constant functions; sine, cosine, and exponential functions; arctangent; hyperbolic sine, cosine, and secant. Products, sums, compositions, derivatives, and antiderivatives of smooth functions are also smooth, as is shown by elementary calculus. In particular, for products of smooth functions, we use *Leibniz' rule* for the n^{th} derivative of a product of two functions:

$$(fg)^{(n)} = \sum_{k=0}^{n} \binom{n}{k} f^{(k)} g^{(n-k)}. \tag{3.9}$$

Here $f^{(0)} \stackrel{\text{def}}{=} f$ and $g^{(0)} \stackrel{\text{def}}{=} g$. Otherwise, $f^{(k)}$ is the k^{th} derivative of f, while $g^{(n-k)}$ is the $(n-k)^{\text{th}}$ derivative of g. The *binomial coefficient* symbol appearing in the sum is the integer coefficient of the x^k term in $(x + 1)^n$, expanded out:

$$\binom{n}{k} \stackrel{\text{def}}{=} \frac{n!}{k!(n-k)!}. \tag{3.10}$$

Smooth local periodization

After localization with a power u^n of the Hanning window of Equation 3.5, the function $u^n f$ satisfies $[u^n f]^{(d)}(0) = [u^n f]^{(d)}(1) = 0$ for $d = 0, \ldots, n$. Hence, its 1-periodic extension will be in $C^d(\mathbf{R})$ by Corollary 3.3. However, a portion of the signal energy within the window is lost, and the operation is not invertible.

If instead we use the wide Hanning window of Equation 3.6, then the 1-periodic function

$$f_1(t) \stackrel{\text{def}}{=} \sum_{n \in \mathbf{Z}} [wf](t+n),$$

where $wf(t) \stackrel{\text{def}}{=} w(t)f(t)$, contains parts of the signal on $[-\frac{1}{2}, \frac{3}{2}]$; the parts from outside $[0,1]$ are said to be *aliased* into the period interval by the summation. Notice that there are at most two nonzero terms in the sum for any t. This also makes the operation noninvertible.

Periodized, the wide Hanning window by itself becomes identically 1 because

$$\sum_{n \in \mathbf{Z}} w(t+n) = \begin{cases} \frac{1}{2}[2 + \sin \pi t + \sin \pi(t+1)], & \text{if } 0 \le t < \frac{1}{2}; \\ \frac{1}{2}[2 + \sin \pi(t-1) + \sin \pi t], & \text{if } \frac{1}{2} \le t < 1, \end{cases} = 1,$$

since $\sin\theta + \sin(\theta \pm \pi) = 0$ for all θ. The Fourier series of this periodization has a single trivial term, namely $1 = \cos 0t$. One consequence is that if f is 1-periodic, then f and the 1-periodization of wf are identical.

Powers of the wide Hanning window do not periodize to the constant function, but there is another way to generalize. Take w as in Equation 3.6, put $w_0 = w$, and for $m > 0$ define

$$w_m(t) = \begin{cases} w_{m-1}(\frac{1}{2}\sin \pi t), & \text{if } t \in [-\frac{1}{2}, \frac{3}{2}], \\ 0, & \text{otherwise.} \end{cases}$$

Using Leibniz' rule, it is possible to show that w_m has 2^m continuous derivatives on $\mathbf{R}$. Figure 3.4 shows w_4 for comparison with Figure 3.2. These smoother wide Hanning windows periodize to the constant function 1 as well: for any $m \ge 0$,

$$\sum_{n \in \mathbf{Z}} w_m(t+n) = 1.$$

There is a third way to localize a function and make it periodic while preserving any smoothness it had originally. This is done by "fraying" the function around the endpoints of an interval, introducing particular boundary conditions, then "splicing" the pieces back together smoothly after periodic extension. It is done in such a way that the aliased portion of the signal from outside the interval cancels itself during periodization, making the operation invertible.

Let $r = r(t)$ be a function in the class $C^d(\mathbf{R})$ for some $0 \le d \le \infty$, satisfying the following conditions:

$$|r(t)|^2 + |r(-t)|^2 = 1 \quad \text{for all } t \in \mathbf{R}; \qquad r(t) = \begin{cases} 0, & \text{if } t \le -1, \\ 1, & \text{if } t \ge 1. \end{cases} \tag{3.11}$$

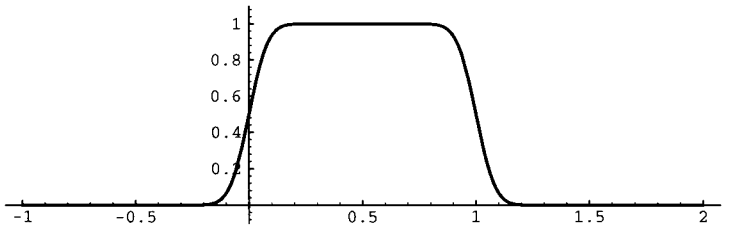


Figure 3.4: Smoother wide Hanning window function with four continuous derivatives.

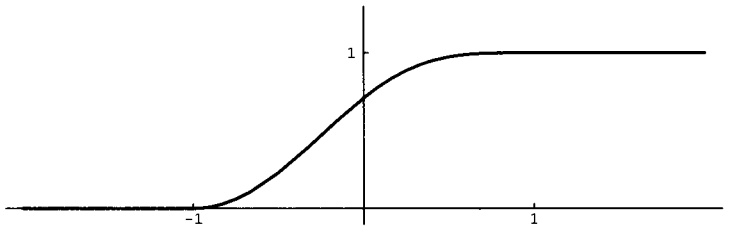


Figure 3.5: Example of a once-differentiable rising cut-off function.

The function r need not be increasing or even real-valued. It is called a *rising cut-off function* because $r(t)$ rises from being identically zero to being identically one as t goes from $-\infty$ to $+\infty$. A general construction for such functions is given in Appendix B. One example of a real-valued, continuous rising cut-off function is the following:

$$r(t) = \begin{cases} 0, & \text{if } t \leq -1, \\ \sin\left[\frac{\pi}{4}(1+t)\right], & \text{if } -1 < t < 1, \\ 1, & \text{if } t \geq 1. \end{cases} \tag{3.12}$$

Smoother functions ($r \in C^d$ for $d > 1$) can be obtained by repeatedly replacing t with $\sin(\pi t/2)$:

$$r_0(t) \stackrel{\text{def}}{=} r(t); \qquad r_{m+1}(t) \stackrel{\text{def}}{=} r_m(\sin\frac{\pi}{2}t). \tag{3.13}$$

Proof by induction may be used to show that $r_m(t)$ has $2^m - 1$ vanishing derivatives at $t = +1$ and $t = -1$, so that $r_m \in C^{2^m-1}$. The case $r_1 \in C^1$ implemented here is depicted in Figure 3.5.

Once-Differentiable Rising Cutoff Function

```
r1( t ):
[0] If t < -1, then return 0
[1] If t >  1, then return 1
[2] Return sin(0.25*PI*(sin(0.5*PI*t)+1.0))
```

Now define a *fraying* operator F and a *splicing* operator S as follows:

$$Fu(t) = \begin{cases} r(t)u(t) + r(-t)u(-t), & \text{if } t > 0, \\ \bar{r}(-t)u(t) - \bar{r}(t)u(-t), & \text{if } t < 0, \\ u(0), & \text{if } t = 0; \end{cases} \tag{3.14}$$

$$Su(t) = \begin{cases} \bar{r}(t)u(t) - r(-t)u(-t), & \text{if } t > 0, \\ r(-t)u(t) + \bar{r}(t)u(-t), & \text{if } t < 0, \\ u(0), & \text{if } t = 0. \end{cases} \tag{3.15}$$

Since $Fu(t) = u(t)$ and $Su(t) = u(t)$ if $t > 1$ or $t < -1$, both may be said to have $[-1, 1]$ as their *reach interval*. There is nothing special about the point 0 or the reach 1. F and S can be moved to the reach interval $[\alpha - \epsilon, \alpha + \epsilon]$ with the formulas of Equation 3.20 further on. The midpoint α is then the *fraying or splicing point*, and $\epsilon \geq 0$ is the *reach*.

Fraying and splicing operators are linear transformations on the vector space of functions on $\mathbf{R}$. If they use the same fixed rising cut-off function r over the same reach interval, then they are inverses. For example, fraying at 0 with reach 1, we have $FSu(0) = u(0)$, $SFu(0) = u(0)$, and for all $t \neq 0$,

$$\begin{aligned} FSu(t) &= \begin{cases} r(t)Su(t) + r(-t)Su(-t), & \text{if } t > 0, \\ \bar{r}(-t)Su(t) - \bar{r}(t)Su(-t), & \text{if } t < 0; \end{cases} \\ &= \begin{cases} r(t)[\bar{r}(t)u(t) - r(-t)u(-t)] \\ \qquad +r(-t)[r(t)u(-t) + \bar{r}(-t)u(t)], & \text{if } t > 0, \\ \bar{r}(-t)[r(-t)u(t) + \bar{r}(t)u(-t)] \\ \qquad -\bar{r}(t)[\bar{r}(-t)u(-t) - r(t)u(t)], & \text{if } t < 0; \end{cases} \\ &= \big(|r(t)|^2 + |r(-t)|^2\big)u(t) \quad = u(t); \end{aligned}$$

$$\begin{aligned} SFu(t) &= \begin{cases} \bar{r}(t)Fu(t) - r(-t)Fu(-t), & \text{if } t > 0, \\ r(-t)Fu(t) + \bar{r}(t)Fu(-t), & \text{if } t < 0; \end{cases} \\ &= \begin{cases} \bar{r}(t)[r(t)u(t) + r(-t)u(-t)] \\ \qquad -r(-t)[\bar{r}(t)u(-t) - \bar{r}(-t)u(t)], & \text{if } t > 0, \\ r(-t)[\bar{r}(-t)u(t) - \bar{r}(t)u(-t)] \\ \qquad +\bar{r}(t)[r(-t)u(-t) + r(t)u(t)], & \text{if } t < 0; \end{cases} \\ &= \big(|r(t)|^2 + |r(-t)|^2\big)u(t) \quad = u(t), \end{aligned}$$

The values $Fu(\alpha)$ and $Su(\alpha)$ at the fraying or splicing point α do not matter in practice since in any implementation using samples of u at integers, we can use $\alpha = -\frac{1}{2}$ to avoid using or defining Fu or Su there. Likewise, in practice we use a nonnegative integer `e` as the reach. For greater generality of implementation, we split the array of samples into its two halves: the samples `uneg[-e]`,...,`uneg[-1]` to the left of the fraying point, and the samples `upos[0]`,...,`upos[e-1]` to the right of the fraying point. The results replace the values in the input arrays:

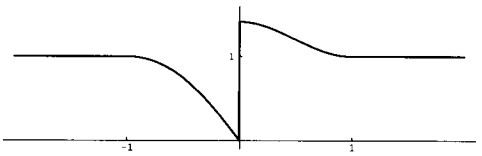


Figure 3.6: Action of fraying on the constant function $u(t) = 1$.

Fraying at a Point Between Two Arrays of Samples

```
fray( uneg[], upos[], e ):
[0] For k=0 to e-1, do [1] to [4]
[1]    Let t = (k+0.5)/e,  let rp = r1(t),  let rn = r1(-t)
[2]    Let up = upos[k],   let un = uneg[-1-k]
[3]    Let upos[  k ] = rp*up + rn*un
[4]    Let uneg[-1-k] = rp*un - rn*up
```

Splicing is implemented in virtually the same way, with just a change of sign in steps 3 and 4:

Splicing at a Point Between Two Arrays of Samples

```
splice( uneg[], upos[], e ):
[0]  For k=0 to e-1, do [1] to [4']
[1]     Let t = (k+0.5)/e,  let rp = r1(t),  let rn = r1(-t)
[2]     Let up = upos[k],   let un = uneg[-1-k]
[3']    Let upos[  k ] = rp*up - rn*un
[4']    Let uneg[-1-k] = rp*un + rn*up
```

Figure 3.6 shows Fu for the example $u(t) \equiv 1$, using the cut-off function r_1 defined in Equation 3.13. Notice how fraying introduces a very particular discontinuity at 0. Splicing takes any function with such a break and repairs it into a smooth function.

Lemma 3.4 *Suppose $r \in C^d(\mathbf{R})$ for some $0 \leq d < \infty$. If $u \in C^d(\mathbf{R})$, then Fu has d continuous derivatives in $\mathbf{R} \setminus \{0\}$, and for all $0 \leq n \leq d$ there exist limits $[Fu]^{(n)}(0+)$ and $[Fu]^{(n)}(0-)$ which satisfy the following conditions:*

$$[Fu]^{(n)}(0+) = 0 \quad \textit{for odd } n; \qquad [Fu]^{(n)}(0-) = 0 \quad \textit{for even } n. \tag{3.16}$$

Conversely, suppose that u belongs to $C^d(\mathbf{R} \setminus \{0\})$ and has limits $u^{(n)}(0+)$ and $u^{(n)}(0-)$ for all $0 \leq n \leq d$ satisfying the equations

$$u^{(n)}(0+) = 0 \quad \textit{for odd } n; \qquad u^{(n)}(0-) = 0 \quad \textit{for even } n. \tag{3.17}$$

Then $Su(0+) = Su(0-)$, and if we define $Su(0) = \lim_{t \to 0} Su(t)$, then Su will belong to $C^d(\mathbf{R})$.

Proof: The smoothness of Fu and Su on $(0,\infty)$ and $(-\infty,0)$ is elementary.

To prove Equation 3.16, calculate the one-sided limits of the derivatives of Fu with Leibniz' rule, Equation 3.9. Since $r^{(n-k)}(0+) = r^{(n-k)}(0-) = r^{(n-k)}(0)$, the resulting expressions factor as follows:

$$[Fu]^{(n)}(0+) = \sum_{k=0}^{n} \binom{n}{k} r^{(n-k)}(0) \left[u^{(k)}(0+) + (-1)^n u^{(k)}(0-)\right]; \tag{3.18}$$

$$[Fu]^{(n)}(0-) = \sum_{k=0}^{n} \binom{n}{k} (-1)^k \overline{r^{(n-k)}(0)} \left[(-1)^n u^{(k)}(0-) - u^{(k)}(0+)\right]. \tag{3.19}$$

If n is odd, with $0 \le n \le d$, then $u^{(k)}(0+) - u^{(k)}(0-) = 0$ in the right-hand side of Equation 3.18 since $u^{(k)}$ is continuous at zero for all $0 \le k \le d$. If n is even, then $u^{(k)}(0-) - u^{(k)}(0+) = 0$ in the right-hand side of Equation 3.19 for the same reason.

To prove that defining $Su(0) = Su(0+)$ gives $Su(t)$ all d continuous derivatives at $t = 0$, use Leibniz' rule and Equation 3.17 to evaluate the following one-sided limits for $0 \le n \le d$:

$$\begin{aligned}
&[Su]^{(n)}(0+) - [Su]^{(n)}(0-) = \\
&\quad = \sum_{k=0}^{n} \binom{n}{k} \Big[\overline{r^{(n-k)}(0)} u^{(k)}(0+) - (-1)^n r^{(n-k)}(0) u^{(k)}(0-) \\
&\qquad - (-1)^{n-k} r^{(n-k)}(0) u^{(k)}(0-) - (-1)^k \overline{r^{(n-k)}(0)} u^{(k)}(0+)\Big] \\
&\quad = \sum_{k=0}^{n} \binom{n}{k} \Big[\left\{1 - (-1)^k\right\} \overline{r^{(n-k)}(0)} u^{(k)}(0+) \\
&\qquad\qquad - (-1)^n \left\{1 + (-1)^k\right\} r^{(n-k)}(0) u^{(k)}(0-)\Big].
\end{aligned}$$

But $\left\{1 - (-1)^k\right\} u^{(k)}(0+) = 0$ and $\left\{1 + (-1)^k\right\} u^{(k)}(0-) = 0$ for all k, so the difference is zero. Thus $\lim_{t\to 0} [Su]^{(n)}(t)$ exists for $0 \le n \le d$, since the one-sided limits agree. Call it $[Su]^{(n)}(0)$. The special case $n = 0$ yields $[Su]^{(0)}(0) = Su(0) = Su(0+) = Su(0-)$.

It remains to show that $[Su]^{(n+1)}(0) = \left([Su]^{(n)}\right)'(0)$ for each $0 \le n < d$. But by the Mean Value Theorem, for each $t \neq 0$ there is some t_0 between 0 and t such that

$$\frac{[Su]^{(n)}(t) - [Su]^{(n)}(0)}{t} = [Su]^{(n+1)}(t_0).$$

Letting $t \to 0$ in this equation yields the result. □

Notice that the constant zero function $f(t) = 0$ satisfies Equation 3.17 at every point.

Fraying may be performed over an arbitrary reach interval $[\alpha - \epsilon, \alpha + \epsilon]$, using the formula:

$$F(r,\alpha,\epsilon)u(t) = \begin{cases} r(\frac{t-\alpha}{\epsilon})u(t) + r(\frac{\alpha-t}{\epsilon})u(2\alpha - t), & \text{if } \alpha < t < \alpha + \epsilon, \\ \bar{r}(\frac{\alpha-t}{\epsilon})u(t) - \bar{r}(\frac{t-\alpha}{\epsilon})u(2\alpha - t), & \text{if } \alpha - \epsilon < t < \alpha, \\ u(t), & \text{otherwise.} \end{cases} \tag{3.20}$$

The formula for $S(r,\alpha,\epsilon)$, or splicing over $[\alpha - \epsilon, \alpha + \epsilon]$, is similar and left as an exercise. It is mostly shown in Equation 3.23 further on.

The boundary conditions at α will be the same as the boundary conditions at zero described in Lemma 3.4. Likewise, splicing over this reach interval undoes the boundary conditions at α. Every $\epsilon > 0$ will yield the same boundary conditions.

Suppose $F_1 = F(r_1,\alpha_1,\epsilon_1)$ and $F_2 = F(r_2,\alpha_2,\epsilon_2)$ are fraying operators with reach intervals $B_1 = [\alpha_1 - \epsilon_1, \alpha_1 + \epsilon_1]$ and $B_2 = [\alpha_2 - \epsilon_2, \alpha_2 + \epsilon_2]$, respectively. If B_1 and B_2 are disjoint, then F_1 and F_2 can be evaluated as follows:

$$F_1F_2u(t) = \begin{cases} F_1u(t), & \text{if } t \in B_1; \\ F_2u(t), & \text{if } t \in B_2; \\ u(t), & \text{otherwise.} \end{cases} \tag{3.21}$$

The same formula may be used to evaluate $F_2F_1u(t)$, so the operators F_1 and F_2 commute. Likewise, splicing operators $S_1 = S(r_1,\alpha_1,\epsilon_1)$ and $S_2 = S(r_2,\alpha_2,\epsilon_2)$ will commute with each other:

$$S_1S_2v(t) = \begin{cases} S_1v(t), & \text{if } t \in B_1; \\ S_2v(t), & \text{if } t \in B_2; \\ v(t), & \text{otherwise,} \end{cases} \quad = S_2S_1v(t). \tag{3.22}$$

Similar formulas show that S_1 commutes with F_2 and S_2 commutes with F_1. The remaining pairs S_1, F_1 and S_2, F_2 commute because they are inverses.

Let $\alpha < \beta$ define an interval, and choose $0 < \epsilon < \frac{1}{2}(\beta - \alpha)$. A smooth function u frayed at $t = \alpha$ and $t = \beta$ with reach intervals $B_\epsilon(\alpha)$ and $B_\epsilon(\beta)$, respectively, may have its ends spliced together with the *loop* operator:

$$\begin{aligned} L(r,[\alpha,\beta],\epsilon)u(t) &= \begin{cases} \bar{r}(\frac{t-\alpha}{\epsilon})u(t) - r(\frac{\alpha-t}{\epsilon})u(2\alpha - t), & \text{if } \alpha < t \le \alpha + \epsilon, \\ r(\frac{\beta-t}{\epsilon})u(t) + \bar{r}(\frac{t-\beta}{\epsilon})u(2\beta - t), & \text{if } \beta - \epsilon \le t < \beta, \\ u(t), & \text{otherwise;} \end{cases} \\ &= \begin{cases} S(r,\alpha,\epsilon)u_I(t), & \text{if } \alpha < t \le \alpha + \epsilon, \\ S(r,\beta,\epsilon)u_I(t), & \text{if } \beta - \epsilon \le t < \beta, \\ u(t), & \text{otherwise.} \end{cases} \end{aligned} \tag{3.23}$$

Here u_I is the periodic extension of u from its localization to I, as defined in Equation 3.7.

The *smooth local periodization* of a function is a combination of fraying at two points and splicing into a loop. Namely, suppose $u = u(t)$ is a smooth function and $I = [\alpha, \beta]$ is an interval. Choose a smooth rising cut-off function r and a positive

reach ϵ satisfying $\epsilon \le \frac{1}{2}(\beta - \alpha) = \frac{1}{2}|I|$. The smooth periodization of u to I is the function

$$(L_I F_\alpha F_\beta u)_I, \tag{3.24}$$

where $F_\alpha = F(r, \alpha, \epsilon)$, $F_\beta = F(r, \beta, \epsilon)$, $L_I = L(r, [\alpha, \beta], \epsilon)$, and the result is localized and periodically extended over the interval I. This is a smooth $|I|$-periodic function.

For example, a smooth locally periodized segment of a sampled signal, with valid sample indices $n \in [a, b) \subset \mathbf{Z}$, may be obtained with the following algorithm:

Local Periodization on $[a, b)$

```
...
[1] Compute   fray( &u[a], &u[a], e )
[2] Compute   fray( &u[b], &u[b], e )
[3] Compute splice( &u[b], &u[a], e )
...
```

Here `&u[a]` signifies the shifted array whose element at index 0 is `u[a]`, and so on. The positive integer reach `e` must be at most half the distance $b - a$, and of course we must have $b > a$. These steps fray u at $\alpha = a - \frac{1}{2}$ and $\beta = b - \frac{1}{2}$, then splice the positive a-end to the negative b-end.

3.1.3 Fourier series

A *Fourier series* is one way to represent a periodic function $f = f(t)$ of one real variable as a superposition of sines and cosines. We start with a fixed set $\{a(0); a(n), b(n), n = 1, 2, \ldots\}$ of *Fourier coefficients* and define f as the limit of *partial sums* $f_N = f_N(t)$:

$$f_N(t) \stackrel{\text{def}}{=} a(0) + \sum_{n=1}^{N} a(n) \left[\sqrt{2}\cos(2\pi n t)\right] + \sum_{n=1}^{N} b(n) \left[\sqrt{2}\sin(2\pi n t)\right]; \tag{3.25}$$

$$f(t) \stackrel{\text{def}}{=} \lim_{N \to \infty} f_N(t), \quad \text{when this limit exists.} \tag{3.26}$$

The finite sum $f_N(t)$ makes sense for every N. In fact it is a smooth 1-periodic function of t, belonging both to **Lip** and C^∞. The limit $f(t)$, on the other hand, may not exist at any t, and even if $f(t)$ exists at every t, the resulting function may not be smooth, or even continuous.

A Fourier series separates f into sine and cosine components with frequencies n and amplitudes $a(n)$ and $b(n)$, respectively. We may find these amplitudes by integration since sines and cosines are orthonormal[2] with respect to the inner product $\langle u, v \rangle = \int_0^1 \bar{u}(t) v(t)\, dt$ on **Lip**. For fixed N, put $\mathbf{F}_N = \{c_0(t) = 1; c_n(t) =$

[2]The strange factor $\sqrt{2}$ is needed to get unit vectors.

$\sqrt{2}\cos(2\pi nt), s_n(t) = \sqrt{2}\sin(2\pi nt) : n = 1, \ldots, N\}$. It is an exercise to show that $\mathbf{F}_N \subset \mathbf{Lip}$ is an orthonormal subset. Since f_N belongs to $\operatorname{span}\mathbf{F}_N \subset \mathbf{Lip}$, the expansion coefficients are obtained by inner products:

$$a(0) = \langle c_0, f_N \rangle ; \qquad a(n) = \langle c_n, f_N \rangle , \quad b(n) = \langle s_n, f_N \rangle , \qquad n = 1, 2 \ldots, N.$$

This establishes a one-to-one correspondence between finite sequences of Fourier coefficients and partial sums of Fourier series:

$$f_N \overset{\mathbf{F}_N}{\longleftrightarrow} \{a(0); a(n), b(n), 1 \leq n \leq N\}.$$

The *energy* in the $a(n)$ or $b(n)$ component over one period is $\|a(n)c_n\|^2 = |a(n)|^2$ or $\|b(n)s_n\|^2 = |b(n)|^2$, respectively, using the norm derived from the inner product. Lemma 2.9 says that these account for all the energy in the partial sum function:

$$\|f_N\|^2 = \int_0^1 |f_N(t)|^2 \, dt = |a(0)|^2 + \sum_{n=1}^{N} \left(|a(n)|^2 + |b(n)|^2 \right). \tag{3.27}$$

Note that for fixed n, $a(n) = \langle c_n, f_N \rangle$ and $b(n) = \langle s_n, f_N \rangle$ for any $N \geq n$. We may therefore construct an orthogonal projection[3] on **Lip** by

$$P_M u \overset{\text{def}}{=} \langle c_0, u \rangle + \sum_{m=1}^{M} \langle c_m, u \rangle \, c_m + \sum_{m=1}^{M} \langle s_m, u \rangle \, s_m, \tag{3.28}$$

which satisfies $P_M f_N = f_M$, for any $M \leq N$. This has the effect of truncating a Fourier series, approximating the limit f with fewer terms.

Any $f \in \mathbf{Lip}$ generates a set of Fourier coefficients:

$$a(0) = \langle c_0, f \rangle = \int_0^1 f(t) \, dt; \tag{3.29}$$

$$a(n) = \langle c_n, f \rangle = \sqrt{2} \int_0^1 f(t) \cos(2\pi nt) \, dt, \tag{3.30}$$

$$b(n) = \langle s_n, f \rangle = \sqrt{2} \int_0^1 f(t) \sin(2\pi nt) \, dt, \tag{3.31}$$

for $n \in \mathbf{Z}^+$. The partial sums then satisfy $f_N = P_N f$ for every N, and for all $0 < M < N$ we have Bessel's inequality: $\|f_M\| \leq \|f_N\| \leq \|f\|$. Therefore, $\sum_{n=1}^{N} |a(n)|^2$ and $\sum_{n=1}^{N} |b(n)|^2$ both converge as $N \to \infty$. We will prove in Lemma 3.5 further on that $\|f - f_N\| \to 0$ as $N \to \infty$, so that we have Parseval's formula for the infinite series:

$$\|f\|^2 = \int_0^1 |f(t)|^2 \, dt = |a(0)|^2 + \sum_{n=1}^{\infty} \left(|a(n)|^2 + |b(n)|^2 \right). \tag{3.32}$$

[3]Why is this a projection? Why is it orthogonal?

These results follow from abstract properties of inner product spaces; see Section 2.3, Exercise 14.

In other words, expanding in the orthonormal system $\mathbf{F} = \{1; s_n, c_n, n \in \mathbf{Z}^+\}$ maps functions in **Lip** to sequences in ℓ^2. The mapping is one-to-one since any $f, g \in \mathbf{Lip}$ giving $\langle f - g, \mathbf{v}\rangle = 0$ for all $v \in \mathbf{F}$ will satisfy $\|f - g\| = 0$ by Parseval's formula. But then $f = g$ since the derived norm is nondegenerate. Hence there is some subset of ℓ^2 which is in one-to-one correspondence with **Lip**. Each Fourier sequence taken from this subset converges to some unique $f \in \mathbf{Lip}$:

$$f \xleftrightarrow{\mathbf{F}} \{a(0), a(1), \ldots; b(1), \ldots\}.$$

There is nothing special about the period 1. If $g = g(t)$ is T-periodic, then $f(t) = \delta_T g(t) \stackrel{\text{def}}{=} \sqrt{T} g(tT)$ is 1-periodic. This δ_T is the so-called *dilation operator*, which is a linear transformation from $\mathbf{Lip}[0, T]$ to $\mathbf{Lip}[0, 1]$, or more generally from functions on $\mathbf{R}$ to dilated functions on $\mathbf{R}$. Thus g is represented by the Fourier series

$$g_N(t) = a(0)\sqrt{\frac{1}{T}} + \sum_{n=1}^{N} a(n)\left[\sqrt{\frac{2}{T}}\cos\left(2\pi\frac{nt}{T}\right)\right] + \sum_{n=1}^{N} b(n)\left[\sqrt{\frac{2}{T}}\sin\left(2\pi\frac{nt}{T}\right)\right],$$

with $g(t) = \lim_{N\to\infty} g_N(t)$ whenever the limit exists. This is an expansion in terms of orthonormal basis functions $\delta_T c_n$ and $\delta_T s_n$. The Fourier coefficients $a(n), b(n)$ are again given by inner products, which in this case are the scaled integrals

$$\begin{aligned}
a(0) &= \langle \delta_T c_0, g\rangle = \sqrt{\frac{1}{T}}\int_0^T g(t)\,dt; \\
a(n) &= \langle \delta_T c_n, g\rangle = \sqrt{\frac{2}{T}}\int_0^T g(t)\cos\left(2\pi\frac{nt}{T}\right)dt, \\
b(n) &= \langle \delta_T s_n, g\rangle = \sqrt{\frac{2}{T}}\int_0^1 g(t)\sin\left(2\pi\frac{nt}{T}\right)dt,
\end{aligned}$$

for $n \in \mathbf{Z}^+$. Because of the factor $\sqrt{T}$, f has the same energy in one period $[0, 1]$ that g has in $[0, T]$, so Parseval's formula applies: $\|g - g_N\| \to 0$ as $N \to \infty$, so

$$\int_0^T |g(t)|^2\,dt = |a(0)|^2 + \sum_{n=1}^{\infty}\left(|a(n)|^2 + |b(n)|^2\right).$$

The integral giving $\|g\|^2$ adds up energy over a single period, which may be considered a time interval. Energy per unit time is called *power*, and thus the listing $\{|a(0)|^2; |a(n)|^2 + |b(n)|^2, n = 1, 2, \ldots\}$ of power by frequency index is called the *power spectrum* of g.

Exponential Fourier series

Another basic 1-periodic function in **Lip** is $e_n = e_n(t) \stackrel{\text{def}}{=} e^{2\pi i n t}$, where n is an integer. This complex-valued function is *unimodular* in the sense that $|e_n(t)| = 1$

for all n and all t. It has uniformly bounded derivative $|e_n'(t)| = |2\pi i n\, e_n(t)| = 2\pi|n|$ for every $t \in \mathbf{R}$, so it belongs to $\mathbf{Lip}(\mathbf{R})$. It is also smooth, with d^{th} derivative $e_n^{(d)}(t) = (2\pi i n)^d e_n(t)$ for every $d \geq 0$ and $t \in \mathbf{R}$, so $e_n \in \mathbf{C}^\infty(\mathbf{R})$ for every $n \in \mathbf{Z}$.

We can use the functions $\{e_n : n \in \mathbf{Z}\}$ to define *exponential Fourier series* from partial sums[4] determined by a sequence $\{c(n) : n \in \mathbf{Z}\} \subset \mathbf{C}$:

$$f_N(t) = \sum_{n=-N}^{N} c(n) e^{2\pi i n t}; \qquad c(n) = \langle e_n, f \rangle = \int_0^1 f(t) e^{-2\pi i n t}\, dt. \tag{3.33}$$

The complex numbers $\{c(n) : n \in \mathbf{Z}\}$ are also called the *(complex exponential) Fourier coefficients* of the function f. We will say that the Fourier series converges if $f(t) = \lim_{N\to\infty} f_N(t)$.

Since $e_n(t) = e^{2\pi i n t} = \cos(2\pi n t) + i \sin(2\pi n t) = \frac{1}{\sqrt{2}} c_n(t) + i \frac{1}{\sqrt{2}} s_n(t)$, the exponential Fourier series is computable from the sine-cosine Fourier series of Equations 3.25, 3.29, 3.30, and 3.31. For $n = 0$, we have $c_0 = 1 = e_0$, and for $n > 0$ we have $c_n(t) = \sqrt{2}\cos(2\pi n t) = [e_n(t) + e_{-n}(t)]/\sqrt{2}$ and $s_n(t) = \sqrt{2}\sin(2\pi n t) = [e_n(t) - e_{-n}(t)]/i\sqrt{2}$. Thus, given $c(n)$ and $c(-n)$ for $n > 0$, we recover $a(n)$ and $b(n)$ by the formulas

$$a(n) = \frac{1}{\sqrt{2}} \langle c_n, f \rangle = \frac{c(n) + c(-n)}{\sqrt{2}}; \tag{3.34}$$

$$b(n) = \frac{1}{\sqrt{2}} \langle s_n, f \rangle = \frac{c(n) - c(-n)}{-i\sqrt{2}} = i\frac{c(n) - c(-n)}{\sqrt{2}}. \tag{3.35}$$

If we define $\{a(0); a(n), b(n), 1 \leq n \leq N\}$ this way from a given sequence $\{c(n) : -N \leq n \leq N\}$, we will have

$$\sum_{n=-N}^{N} c(n) e^{2\pi i n t} = a(0) + \sum_{n=1}^{N} a(n) \left[\sqrt{2}\cos(2\pi n t)\right] + \sum_{n=1}^{N} b(n) \left[\sqrt{2}\sin(2\pi n t)\right]. \tag{3.36}$$

Conversely, for $n > 0$,

$$c(n) = \frac{1}{\sqrt{2}} \langle c_n + i s_n, f \rangle = \frac{a(n) - i b(n)}{\sqrt{2}}. \tag{3.37}$$

Since $-n < 0$ and $\sin(2\pi[-n]t) = -\sin(2\pi n t)$ and $\cos(2\pi[-n]t) = \cos(2\pi n t)$, the negative-indexed coefficients satisfy

$$c(-n) = \frac{1}{\sqrt{2}} \langle c_n - i s_n, f \rangle = \frac{a(n) + i b(n)}{\sqrt{2}}. \tag{3.38}$$

The remaining case of $n = 0$ gives $c(0) = \langle 1, f \rangle = a(0)$. If we define $\{c(n) : -N \leq n \leq N\}$ in this way from given $\{a(0); a(n), b(n), 1 \leq n \leq N\}$, we will again have Equation 3.36.

[4] Reuse of the notation f_N is justified by Equation 3.36.

With matrices, for $n > 0$ we get

$$\begin{pmatrix} c(n) \\ c(-n) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ 1 & i \end{pmatrix} \begin{pmatrix} a(n) \\ b(n) \end{pmatrix}; \quad \begin{pmatrix} a(n) \\ b(n) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix} \begin{pmatrix} c(n) \\ c(-n) \end{pmatrix}.$$

It is not hard to check that these matrices are unitary.

Complex exponential Fourier coefficients are computed via the formula

$$c(n) = \langle e_n, f \rangle = \int_0^1 f(t) e^{-2\pi i n t} \, dt \stackrel{\text{def}}{=} \hat{f}(n). \tag{3.39}$$

We can regard the computation $f \mapsto \hat{f}$ as a transformation from functions to sequences. It is a linear transformation with a domain containing **Lip**. By Parseval's formula, the range is a subset of ℓ^2.

The underlying functions superpose neatly to give everything in **Lip**:

Lemma 3.5 *The functions* $\{e_n = e_n(t) \stackrel{\text{def}}{=} e^{2\pi i n t} : n \in \mathbf{Z}\}$ *are an orthonormal basis for* **Lip**.

Proof: For orthogonality, note that if n, m are integers, then

$$\begin{aligned} \langle e_n, e_m \rangle &= \int_0^1 \overline{e^{2\pi i n t}} e^{2\pi i m t} \, dt = \int_0^1 e^{2\pi i (m-n) t} \, dt \\ &= \begin{cases} \int_0^1 1 \, dt = 1, & \text{if } n = m; \\ \left. \frac{e^{2\pi i (m-n) t}}{2\pi i (m-n)} \right|_0^1 = 0, & \text{if } n \neq m. \end{cases} \end{aligned}$$

For completeness, it suffices to show that an arbitrary 1-periodic piecewise linear function $f \in X$ satisfies $\|f - f_N\| \to 0$ as $N \to \infty$, where f_N is the partial sum of f's exponential Fourier series, defined by Equation 3.33. But it is enough to show this for the general hat function determined by $0 \leq a < b < c \leq 1$:

$$f(t) = \begin{cases} 0, & \text{if } t < a \text{ or } t > c, \\ (t-a)/(b-a), & \text{if } a \leq t \leq b, \\ (c-t)/(c-b), & \text{if } b \leq t \leq c. \end{cases}$$

That is because all piecewise linear functions are linear combinations of such f. The explicit approximation is left as an exercise. □

If $e_n(t)$ is replaced with $e_n(t/T)/\sqrt{T}$ for each n, then the same formulas work for T-periodic functions.

In fact, the complex exponentials superpose to give all square-integrable functions, although in that case the Fourier series may not converge at every point[5] in the domain $[0, 1]$. Bessel's inequality, Parseval's formula, and Lemma 3.5 may be combined to yield the following, which is proved as Theorem 11.4, page 309 of Apostol's *Mathematical Analysis*:

[5] In 1966, Lennart Carleson showed that the Fourier series of a square-integrable function converges at *almost* every point in the domain.

Corollary 3.6 *A function f belongs to $L^2([0,1])$ if and only if the Fourier coefficients $\{c(n) : n \in \mathbf{Z}\}$ for f belong to ℓ^2. In that case, the partial sums f_N of f's Fourier series satisfy $\lim_{N\to\infty} \|f - f_N\| = 0$, with the estimate $\|f - f_N\|^2 = \sum_{|n|>N} |c(n)|^2$.* □

Suppose we reverse the construction. Starting with an infinite sequence $c = \{c(n) : n \in \mathbf{Z}\} \in \ell^2$, let us define its *inverse Fourier transform* to be the 1-periodic function given by the following infinite sum:

$$\check{c}(x) \stackrel{\text{def}}{=} \sum_{k=-\infty}^{\infty} c(k) e^{2\pi i k x}. \tag{3.40}$$

Then $c \mapsto \check{c}$ is a linear transformation from sequences to 1-periodic functions. By Corollary 3.6, its domain is all of ℓ^2 and its range is $L^2([0,1])$. But there is a simple condition on c that suffices to guarantee $\check{c} \in \mathbf{Lip}$:

Theorem 3.7 *If there is some fixed constant M such that the sequence $c = \{c(k) : k \in \mathbf{Z}\}$ of complex numbers satisfies*

$$\sum_{k=-N}^{N} |k|\,|c(k)| \le M, \qquad \text{all integers } N, \tag{3.41}$$

then

a. *the partial sums $\check{c}_N(x) \stackrel{\text{def}}{=} \sum_{k=-N}^{N} c(k) e^{2\pi i k x}$ converge absolutely to the inverse Fourier transform $\check{c}(x) \stackrel{\text{def}}{=} \lim_{N\to\infty} \check{c}_N(x)$ of c;*

b. *the function $\check{c}$ belongs to* **Lip**, *with $|\check{c}(x) - \check{c}(y)| \le 2\pi M |x - y|$, for all $x, y \in \mathbf{R}$.*

Proof: For (a), note that at any $x \in \mathbf{R}$, the Fourier series with coefficients c converges absolutely:

$$\sum_{k=-N}^{N} |c(k) e^{2\pi i k x}| = \sum_{k=-N}^{N} |c(k)| \le |c(0)| + \sum_{k=-N}^{N} |k|\,|c(k)| \le |c(0)| + M,$$

so since these sums increase with N and are bounded above, they must converge to a limit no greater than $|c(0)| + M$, and independent of x. Absolute convergence of a series implies convergence, so the Fourier series $\check{c}(x)$ converges at every x.

For (b), observe that for any $t \in \mathbf{R}$,

$$|e^{it} - 1| = \left| i \int_0^t e^{ix}\, dx \right| \le \int_0^t |e^{ix}|\, dx = |t|.$$

Thus $|e^{2\pi i k x} - e^{2\pi i k y}| = |e^{2\pi i k (x-y)} - 1| \le 2\pi |k|\,|x - y|$, for any $x, y \in \mathbf{R}$ and any integer k. That imposes a Lipschitz condition on the partial sum $\check{c}_N$ at any

$x, y \in \mathbf{R}$:

$$\begin{aligned} |\check{c}_N(x) - \check{c}_N(y)| &= \left| \sum_{k=-N}^{N} c(k)[e^{2\pi ikx} - e^{2\pi iky}] \right| \\ &\leq 2\pi|x-y| \sum_{k=-N}^{N} |k|\,|c(k)| \leq 2\pi M|x-y|. \end{aligned}$$

But then by part (a), we may take limits as $N \to \infty$ to get $|\check{c}(x)-\check{c}(y)| \leq 2\pi M|x-y|$. □

Smoothness and rapid decrease

If $f = f(x)$ is differentiable on $[0,1]$, then integration by parts gives the following formula for its (complex exponential) Fourier coefficients:

$$\begin{aligned} \widehat{f'}(n) &= \int_0^1 f'(x)e^{-2\pi inx}\,dx = \int_0^1 e^{-2\pi inx}\,df(x) \\ &= \left[e^{-2\pi inx}f(x)\right]_0^1 + (2\pi in)\int_0^1 f(x)e^{-2\pi inx}\,dx \\ &= f(1) - f(0) + (2\pi in)\,\hat{f}(n). \end{aligned}$$

Using induction, this extends to a formula for the Fourier coefficients of derivative $d+1$ in terms of the d^{th} derivative:

$$\widehat{f^{(d+1)}}(n) = f^{(d)}(1) - f^{(d)}(0) + (2\pi in)\,\widehat{f^{(d)}}(n). \tag{3.42}$$

The first application is an estimate for the rate of decrease of Fourier coefficients of a function in terms of its degree of smoothness:

Lemma 3.8 *Suppose that $f^{(d)}$ is continuous and 1-periodic for some integer $d \geq 0$. Then $\hat{f}(n) = O(|n|^{-d})$ as $n \to \pm\infty$.*

Proof: If f is 1-periodic and has a continuous d^{th} derivative on all of $\mathbf{R}$, then $f^{(k)}(1) = f^{(k)}(0)$ for all $k = 0, 1, \ldots, d$, so we may iterate Equation 3.42 without the boundary terms to get $\widehat{f^{(d)}}(n) = (2\pi in)^d \hat{f}(n)$. But the Fourier coefficients of any continuous function g are bounded:

$$|\hat{g}(n)| = \left| \int_0^1 g(x)e^{-2\pi inx}\,dx \right| \leq \int_0^1 |g(x)e^{-2\pi inx}|\,dx = \int_0^1 |g(x)|\,dx < \infty, \tag{3.43}$$

independently of the frequency n. Putting $g = f^{(d)}$ shows that the quantity $|2\pi n|^d|\hat{f}(n)|$ must remain bounded as $n \to \infty$, so $\hat{f}(n) = O(|n|^{-d})$. □

In other words, the more derivatives a periodic function has, the faster its Fourier coefficients tend to zero with increasing frequency. This principle applies to both

sine-cosine and exponential Fourier series. Consequently, if the Fourier series of a smooth function is truncated to a finite partial sum involving only frequencies $|n| \leq N$, then the resulting error will tend to zero as $N \to \infty$ at a rate like[6]

$$|f(x) - f_N(x)| = O\left(\sum_{|n|>N} |n|^{-d}\right) = O(N^{-d+1}),$$

which we obtain using the integral test to sum the series. The smoother the function, the faster the error will go away or, from another perspective, the fewer terms will be needed for a desired accuracy.

Recall that if $f \in \mathbf{Lip}$ satisfies $f(0) = f(1)$, then f has a continuous 1-periodic extension which satisfies the Lipschitz condition on all of $\mathbf{R}$. That in fact is enough to guarantee that the Fourier series for $f(x)$ converges at every $x \in [0,1]$, though we get no error estimate. The proof is beyond our scope, but may be found in Apostol as Theorem 11.9 on page 316.

Fourier coefficients can be computed over any period interval $[a, a+T]$:

$$\hat{f}(k) = \frac{1}{\sqrt{T}} \int_a^{a+T} f(t) e^{-2\pi i k t/T}\, dt.$$

They thus depend on f's behavior as a periodic function on the whole line, not just in the interior of one period interval such as $[0,T]$. In particular, discontinuities hiding at the endpoints of a period interval can affect the rate of decrease of Fourier coefficients because they can affect the cancellation of the boundary terms in Equation 3.42.

Even without any derivatives, an integrable function f must have some decrease in its Fourier coefficients $\hat{f}(n)$, as $n \to \pm\infty$, although we get no rate estimate:

Lemma 3.9 (Riemann–Lebesgue) *If $f \in \mathbf{Lip}$, then $\hat{f}(k) \to 0$ as $|k| \to \infty$.*

Proof: For simplicity of notation, we will denote the 1-periodic extension of f by f as well. We will write $C > 0$ for the constant in the Lipschitz condition, so $|f(x) - f(y)| \leq C|x - y|$ for all $x, y \in [0,1]$. The 1-periodic extension might not satisfy this condition, but there is still some constant $M > 0$ such that $|f(x)| \leq M$ for all $x \in [0,1]$, hence also for all $x \in \mathbf{R}$, so $|f(x) - f(y)| \leq 2M$ for all $x, y \in \mathbf{R}$.

Now $\hat{f}(k)$ may be computed in two ways, using the fact that $e^{\pm i\pi} = -1$:

$$\begin{aligned}
\int_0^1 f(t) e^{-2\pi i k t}\, dt &= \int_0^1 f\left(t + \frac{1}{2k}\right) e^{-2\pi i k t - i\pi}\, dt \\
&= -\int_0^1 f\left(t + \frac{1}{2k}\right) e^{-2\pi i k t}\, dt \\
\Rightarrow\ 2\hat{f}(k) &= \int_0^1 \left[f(t) - f\left(t + \frac{1}{2k}\right)\right] e^{-2\pi i k t}\, dt
\end{aligned}$$

[6]The estimated, or worst-case error, is independent of x, though of course for certain x it could be much smaller.

$$\Rightarrow\ 2|\hat{f}(k)| \ \leq\ \int_0^1 \left| f(t) - f\left(t + \frac{1}{2k}\right)\right| dt.$$

Thus, we will have $|f(t) - f(t + \frac{1}{2k})| \leq \frac{C}{2|k|}$ for every $k \neq 0$, as long as both $t \in [0,1]$ and $t + \frac{1}{2k} \in [0,1]$. Now suppose $\epsilon > 0$ is given. We choose $\frac{1}{2} > \delta > 0$ such that $\delta < \frac{\epsilon}{4M}$, and we put $K = \max\{\frac{1}{2\delta}, \frac{C}{2\epsilon}\}$. Then $|k| > K$ will guarantee $t \in [\delta, 1-\delta] \Rightarrow (t + \frac{1}{2k}) \in [0,1]$, and also $\frac{C}{2|k|} < \epsilon$. Hence, the Fourier coefficient $\hat{f}(k)$ may be estimated as follows:

$$\begin{aligned} 2|\hat{f}(k)| \ &\leq\ \int_0^\delta 2M\, dt + \int_\delta^{1-\delta} \frac{C}{2|k|}\, dt + \int_{1-\delta}^1 2M\, dt \\ &<\ 2M\delta + (1-2\delta)\epsilon + 2M\delta \ <\ 2\epsilon. \end{aligned}$$

Since ϵ was arbitrary, it follows that $|\hat{f}(k)| \to 0$ as $|k| \to \infty$. □

More careful analysis shows that the Lipschitz continuity hypothesis may be replaced by simple continuity on $[0,1]$, giving a strict improvement over Equation 3.43. In fact just absolute integrability of f on $[0,1]$ is enough. The same proof works, but the last step is to show that $\int_0^1 |f(t) - f(t+x)|\, dt \to 0$ as $x \to 0$ for any absolutely integrable f. That, however, is beyond the scope of this text.

The converse problem is to estimate the number of continuous derivatives of a function, given just the rate of decrease of its Fourier coefficients. That is an easy generalization of Theorem 3.7:

Corollary 3.10 *If $d \geq 0$ and there is some fixed constant M such that the sequence $c = \{c(k) : k \in \mathbf{Z}\}$ of complex numbers satisfies*

$$\sum_{k=-N}^{N} |k|^{d+1}\, |c(k)| \leq M, \qquad \textit{all integers } N, \tag{3.44}$$

then

a. *the partial sums $\check{c}_N(x) \stackrel{\text{def}}{=} \sum_{k=-N}^{N} c(k) e^{2\pi i k x}$ converge absolutely to the inverse Fourier transform $\check{c}(x) \stackrel{\text{def}}{=} \lim_{N\to\infty} \check{c}_N(x)$ of c;*

b. *the function $f = f(x) \stackrel{\text{def}}{=} \check{c}(x)$ may be differentiated d times, and $f^{(d)} \in \mathbf{Lip}$, with $|f^{(d)}(x) - f^{(d)}(y)| \leq 2\pi M |x - y|$, for all $x, y \in \mathbf{R}$.* □

A *finite Fourier series*, or *trigonometric polynomial*, results if $\hat{f}(k) = 0$ for all sufficiently large $|k|$. This is just a linear combination of sines and cosines, so it has a d^{th} derivative for every d. Similarly, if the Fourier coefficients of f satisfy $\hat{f}(k) \leq Cr^{|k|}$ for $r, C > 0$ with $r < 1$, then $(2\pi i k)^d \hat{f}(k) \to 0$ as $|k| \to \infty$ for every d, and we may conclude that f is smooth.

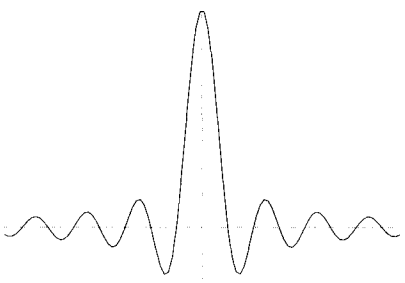


Figure 3.7: Part of the graph of sinc (ξ).

Fourier integrals

If $u = u(x)$ is a complex-valued function of one real variable x, the *Fourier integral transform* of u is another complex-valued function, denoted $\mathcal{F}u = \mathcal{F}u(\xi)$, depending on one real variable ξ, defined by the following integral:

$$\mathcal{F}u(\xi) = \int_{-\infty}^{\infty} e^{-2\pi i x\xi} u(x)\, dx, \tag{3.45}$$

whenever this improper integral converges.

It is possible to compute the Fourier integral transform of certain simple functions exactly. For example, let $u = \mathbf{1}_I$ be the characteristic function of the balanced unit interval $I = [-\frac{1}{2}, \frac{1}{2}]$:

$$\mathbf{1}_I(x) = \begin{cases} 1, & \text{if } -\frac{1}{2} \le x \le \frac{1}{2}; \\ 0, & \text{otherwise.} \end{cases} \tag{3.46}$$

An elementary calculation shows that if $\xi \neq 0$, then

$$\mathcal{F}\mathbf{1}_I(\xi) = \int_{-\infty}^{\infty} \mathbf{1}_I(x) e^{-2\pi i x\xi}\, dx = \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{-2\pi i x\xi}\, dx = \frac{e^{-\pi i\xi} - e^{\pi i\xi}}{-2\pi i\xi} = \frac{\sin(\pi\xi)}{\pi\xi}.$$

If we put $\mathcal{F}\mathbf{1}_I(0) \stackrel{\text{def}}{=} 1$ for continuity, the resulting function is called *sinc*:

$$\operatorname{sinc}(\xi) \stackrel{\text{def}}{=} \begin{cases} \frac{\sin(\pi\xi)}{\pi\xi}, & \text{if } \xi \neq 0; \\ 1, & \text{if } \xi = 0. \end{cases} \tag{3.47}$$

A similar calculation, left as an exercise, shows that the Fourier integral transform of $\mathbf{1}_I(x/T)$, which is the characteristic function of $[-\frac{T}{2}, \frac{T}{2}]$, is $T\operatorname{sinc}(T\xi) = \frac{\sin(T\pi\xi)}{\pi\xi}$. Figure 3.7 shows part of the graph of sinc; notice the many local maxima to either side of the global maximum at 0, and the zero-crossings at all other integers.

The *Gaussian function* $g(x) = e^{-\pi x^2}$ is another function whose Fourier integral transform can be computed exactly. We start with the fact[7] that $\int_{\mathbf{R}} e^{-\pi x^2}\,dx = 1$ and change variables:

$$\begin{aligned} 1 &= \int_{-\infty}^{\infty} e^{-\pi y^2}\,dy = \int_{-\infty}^{\infty} e^{-\pi(x+i\xi)^2}\,dx \\ &= e^{\pi\xi^2}\int_{-\infty}^{\infty} e^{-2\pi i x\xi} e^{-\pi x^2}\,dx = e^{\pi\xi^2}\mathcal{F}g(\xi), \end{aligned} \tag{3.48}$$

where we make the substitutions $y \leftarrow x + i\xi$ and $dy = dx$.[8] Multiplying by $g(\xi) = e^{-\pi\xi^2}$ shows that $\mathcal{F}g = g$.

The Fourier integral transform can be applied to any square-integrable function, although the proof of this is beyond our scope.[9] It may therefore be applied to any vector in the space $L^2(\mathbf{R})$, whose inner product and derived norm are given in Equations 2.29 and 2.24, respectively. In fact, $\mathcal{F} : L^2(\mathbf{R}) \to L^2(\mathbf{R})$ is a unitary transformation:

Theorem 3.11 (Plancherel) *Suppose $u = u(t)$ and $v = v(t)$ are square-integrable functions, and let $\langle u, v\rangle \stackrel{\text{def}}{=} \int \bar{u}(t)v(t)\,dt$. Then $\mathcal{F}u$ and $\mathcal{F}v$ are square-integrable and satisfy $\langle \mathcal{F}u, \mathcal{F}v\rangle = \langle u, v\rangle$.* □

A sketch of the proof may be found in Section B.6 of the appendix. We may expand the integrals and change their order to get another useful result:

Corollary 3.12 *If $u = u(t)$ and $v = v(t)$ are square-integrable functions, then*

$$\int_{-\infty}^{\infty} \bar{u}(t)v(t)\,dt = \iiint_{-\infty}^{\infty} e^{2\pi i\xi(x-y)}\bar{u}(x)v(y)\,dx dy d\xi.$$

Proof: By Plancherel's theorem,

$$\begin{aligned} \int_{-\infty}^{\infty} \bar{u}(t)v(t)\,dt &= \langle u, v\rangle = \langle \mathcal{F}u, \mathcal{F}v\rangle \\ &= \int_{-\infty}^{\infty} \left(\overline{\int_{-\infty}^{\infty} e^{-2\pi i x\xi}u(x)\,dx}\right)\left(\int_{-\infty}^{\infty} e^{-2\pi i y\xi}v(y)\,dy\right)d\xi, \end{aligned}$$

and the complex conjugation passes through to the first integrand. □

The *inverse Fourier integral transform* of a square-integrable function $v = v(\xi)$ may thus be determined from the formula $\langle u, v\rangle = \langle \mathcal{F}u, \mathcal{F}v\rangle = \langle u, \mathcal{F}^*\mathcal{F}v\rangle$, which holds

[7] This is proved as Equation B.6 in Appendix B, page 286.

[8] See Apostol's Theorem 16.12, page 424 for a rigorous version of this informal calculation.

[9] See Stein and Weiss, *Introduction to Fourier Analysis on Euclidean Spaces*, Chapter 1, for a nice exposition of the theory of Fourier integrals.

for all $u \in L^2(\mathbf{R})$. From the nondegeneracy of the inner product, we conclude that $\mathcal{F}^{-1} = \mathcal{F}^*$, and this may be calculated by interchanging the order of integration:

$$\begin{aligned} \langle u, \mathcal{F}^* v \rangle &= \langle \mathcal{F}u, v \rangle = \int_{-\infty}^{\infty} \left(\overline{\int_{-\infty}^{\infty} e^{-2\pi i x\xi} u(x)\, dx} \right) v(\xi)\, d\xi \\ &= \int_{-\infty}^{\infty} \bar{u}(x) \left(\int_{-\infty}^{\infty} e^{2\pi i t x} v(\xi)\, d\xi \right) dx. \end{aligned}$$

Identifying the integrand factor other than $\bar{u}$ with $\mathcal{F}^* v$ gives

$$\mathcal{F}^{-1} v(x) = \mathcal{F}^* v(x) = \int_{-\infty}^{\infty} e^{2\pi i x\xi} v(\xi)\, d\xi, \tag{3.49}$$

where $v = v(\xi)$ is any square-integrable complex-valued function of the real variable ξ.

$\mathcal{F}$ and $\mathcal{F}^{-1} = \mathcal{F}^*$ are practically the same transform. A change of variable in the integrands shows that $\mathcal{F}u(x) = \mathcal{F}^{-1}u(-x)$, so in particular for the balanced interval $I = [-\frac{1}{2}, \frac{1}{2}]$, we have

$$\mathcal{F}^{-1}\mathbf{1}_I(x) = \mathcal{F}\mathbf{1}_I(-x) = \operatorname{sinc}(-x) = \operatorname{sinc}(x),$$

since sinc is symmetric with respect to $x \mapsto -x$. Thus sinc is also the inverse Fourier integral transform of $\mathbf{1}_I$. Likewise, the symmetric function $\mathbf{1}_I$ is both $\mathcal{F}^{-1}\mathrm{sinc}$ and $\mathcal{F}\mathrm{sinc}$, and the symmetric Gaussian function $g(x) = e^{-\pi x^2}$ satisfies $g = \mathcal{F}g = \mathcal{F}^{-1}g$.

3.2 Discrete Fourier Analysis

To compute Fourier coefficients requires evaluating integrals of rapidly varying functions: the factor $e^{2\pi i k x}$ oscillates rapidly when $|k|$ is large. In rare cases, such integrals can be computed analytically by calculus, but the Fourier transform is far too useful to be restricted that way. For the majority of cases, numerical integration must be used.

To do that, fix a large integer N and put $t_n = n/N$ for $n = 0, 1, \ldots, N$. These form a grid of $N+1$ equally spaced points in the interval $[0, 1]$, with $\Delta t_n \stackrel{\text{def}}{=} t_{n+1} - t_n = \frac{1}{N}$ for all k. Now suppose that $u = u(t)$ is a 1-periodic function that is to be approximated by its Fourier series at the points $\{t_n\}$. Since $u(t_0) = u(t_N)$, the Riemann sum approximating the integral for the k^{th} Fourier coefficient may be taken over just $n = 0, 1, \ldots, N-1$:

$$\hat{u}(k) \approx p_k \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} u(t_n)\, e^{-2\pi i k t_n}\, \Delta t_n = \frac{1}{N} \sum_{n=0}^{N-1} u\left(\frac{n}{N}\right) \exp\left(-2\pi i \frac{kn}{N}\right), \tag{3.50}$$

for $k \in \mathbf{Z}$. The approximation improves as $N \to \infty$, that is, with a denser sampling of u on $[0, 1]$.

Notice that $\{p_k : k \in \mathbf{Z}\}$ is actually an N-periodic sequence since

$$\begin{aligned} p_{k+N} &= \exp\left(-2\pi i\frac{(k+N)n}{N}\right) = \exp(-2\pi in)\exp\left(-2\pi i\frac{kn}{N}\right) \\ &= \exp\left(-2\pi i\frac{kn}{N}\right) = p_k. \end{aligned} \tag{3.51}$$

For fixed N, the computation $u \mapsto p$ defined by Equation 3.50, which maps a 1-periodic function to an N-periodic sequence, is called the N-point *discrete Fourier transform*, or *DFT*.

Computing the DFT using N sample points is equivalent to applying an $N \times N$ matrix to a vector. For a general matrix, this costs $O(N^2)$ arithmetic operations and becomes burdensome rather rapidly as the number of sample points increases. However, we will see how the Fourier matrix can be factored into a product of just a few sparse matrices, and the resulting factors can be applied to a vector in a total of $O(N \log N)$ arithmetic operations, lowering the cost for large N. This factored, speedier implementation is the so-called *fast Fourier transform*, or *FFT*.

3.2.1 Discrete Fourier transform

To fix notation, let $v \in \mathbf{C}^N$ be a vector, indexed as $v = \{v(n)\}_{n=0}^{N-1}$. The discrete Fourier transform of v is the vector $\hat{v} \in \mathbf{C}^N$ defined by

$$\hat{v}(m) = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} v(n)\exp\left(-2\pi i\frac{mn}{N}\right),$$

for $m = 0, 1, \ldots, N-1$. The factor $\frac{1}{\sqrt{N}}$ normalizes the output and makes the transform unitary.

The matrix form of this equation is $\hat{v} = \frac{1}{\sqrt{N}}Fv$, where $F : \mathbf{C}^N \to \mathbf{C}^N$ is defined by

$$F(m,n) = \exp\left(-2\pi i\frac{mn}{N}\right). \tag{3.52}$$

The subscript N in F_N will be added when necessary to emphasize the dimension of the range and domain. Also, writing $\omega_n = \exp\left(-2\pi i\frac{n}{N}\right)$ gives

$$F = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_1 & \omega_1^2 & \cdots & \omega_1^{N-1} \\ 1 & \omega_2 & \omega_2^2 & \cdots & \omega_2^{N-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_{N-1} & \omega_{N-1}^2 & \cdots & \omega_{N-1}^{N-1} \end{pmatrix}; \qquad F(m,n) = (\omega_m^n)_{m,n=0}^{N-1}. \tag{3.53}$$

Such a matrix is called a *Vandermonde* matrix: its m^{th} row is the successive powers $\{1, \omega_m, \omega_m^2, \ldots\}$ of ω_m. Induction on N proves that

$$\det F = \prod_{0 \le n < m < N} (\omega_m - \omega_n),$$

and since $\omega_n \neq \omega_m$ for $n \neq m$, F_N is nonsingular. Also, $\omega_n^m = \omega_m^n$, so F is a symmetric matrix; $\bar{\omega}_m = \omega_m^{-1}$, since $|\omega_m| = 1$; and $\omega_m^N = 1$ for all $m = 0, 1, \ldots, N-1$. Thus $\frac{1}{\sqrt{N}}F$ is unitary:

$$\begin{aligned} FF^*(m,n) &= \sum_{k=0}^{N-1} \omega_m^k \bar{\omega}_n^k = \sum_{k=0}^{N-1} \omega_1^{(m-n)k} = \begin{cases} N, & \text{if } m = n, \\ \frac{1-\omega_1^{(m-n)N}}{1-\omega_1^{m-n}} = 0, & \text{if } m \neq n, \end{cases} \\ &= N\delta(m-n), \qquad \Rightarrow \frac{1}{\sqrt{N}}F\left[\frac{1}{\sqrt{N}}F\right]^* = \frac{1}{N}FF^* = Id. \end{aligned}$$

The formula for $\hat{v}(m)$ makes sense at any integer m, and satisfies $\hat{v}(m+N) = \hat{v}(m)$ for every m, since $\exp(-2\pi i(m+N)n/N) = \exp(-2\pi imn/N)$ for every integer n. Thus, the discrete Fourier coefficients are N-periodic and no additional information is contained in them beyond the first N. Since $\hat{v}(-m) = \hat{v}(N-m)$, the discrete Fourier coefficients $\hat{v}([N-1]/2), \ldots, \hat{v}(N-1)$ duplicate the Fourier coefficients $\hat{v}(-[N-1]/2), \ldots, \hat{v}(-1)$ at negative frequencies, when N is odd. When N is even, it is coefficients $\hat{v}(N/2), \hat{v}(N/2+1), \ldots, \hat{v}(N-1)$ that may be regarded as duplicates of the negative-frequency coefficients $\hat{v}(-N/2), \hat{v}(-N/2+1), \ldots, \hat{v}(-1)$. Note the similarity with the division of bit strings into positive and negative integers for twos-complement arithmetic, shown in Figure 1.1.

Since $v = \frac{1}{\sqrt{N}}F^*\hat{v}$, it lies in the column space of the adjoint matrix F^*. These columns are the *discrete Fourier basis functions* $\exp\left(2\pi i\frac{mn}{N}\right)$. Figure 3.8 shows the real and imaginary parts of an example with $N = 256$ and $m = 3$. Notice that with so many sample points it is difficult to distinguish the graph from that of a smooth function.

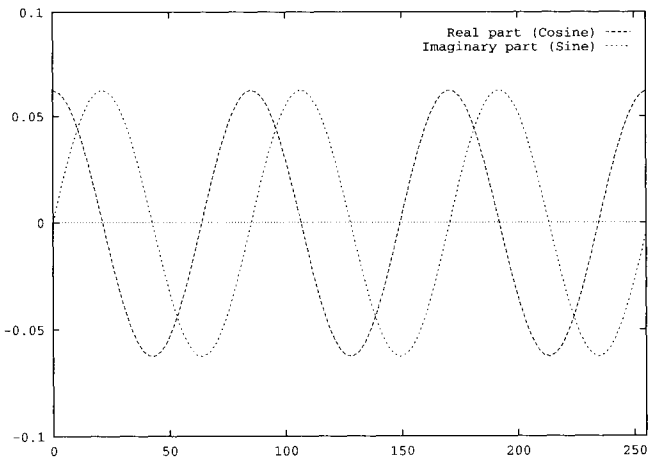


Figure 3.8: Real and imaginary parts of a Fourier basis function $\frac{1}{16}\exp\left(2\pi i\frac{3n}{256}\right)$.

The "fast" discrete Fourier transform

Every coefficient of the matrix $F(m,n)$ has absolute value one, so F is not sparse. However, algebraic properties of the exponential function may be exploited to reduce its cost of application, using the following:

Lemma 3.13 (Danielson-Lanczos, 1942) *Suppose that N is an even positive integer. Then the N-point discrete Fourier transform F_N splits into two parts: for $m = 0, 1, \ldots, N-1$,*

$$F_N v(m) = F_{\frac{N}{2}} v^e(m) + \exp(-2\pi i m/N)\, F_{\frac{N}{2}} v^o(m).$$

Here $v^e(n) \stackrel{\text{def}}{=} v(2n)$ and $v^o(n) \stackrel{\text{def}}{=} v(2n+1)$ for $0 \le n < \frac{N}{2}$, and the two $\frac{N}{2}$-point discrete Fourier transforms $F_{\frac{N}{2}}$ are considered $\frac{N}{2}$-periodic for purposes of evaluation at $\frac{N}{2} \le m < N$.

Proof: For any $0 \le m < N$, the sum in n defining $F_N v(m)$ splits into the even-n and odd-n parts:

$$\begin{aligned}
F_N v(m) &= \sum_{n=0}^{N-1} v(n) \exp(-2\pi i m n/N) \\
&= \sum_{n=0}^{N/2-1} v(2n) \exp(-2\pi i (2n) m/N) \\
&\qquad + \sum_{n=0}^{N/2-1} v(2n+1) \exp(-2\pi i (2n+1) m/N) \\
&= \sum_{n=0}^{N/2-1} v(2n) \exp\left(\frac{-2\pi i n m}{N/2}\right) \\
&\qquad + \exp(-2\pi i m/N) \sum_{n=0}^{N/2-1} v(2n+1) \exp\left(\frac{-2\pi i n m}{N/2}\right).
\end{aligned}$$

The two sums are evidently $F_{\frac{N}{2}} v^e(m)$ and $F_{\frac{N}{2}} v^o(m)$. □

What does this achieve? Since the number of operations needed to compute $F_N v$ equals the total number of nonzero matrix coefficients, F_N costs N^2 while the split version costs $\frac{1}{2}N^2$ plus about $3N$ for preparing v^e and v^o and later reassembling the pieces. Thus the split version is cheaper for $N > 6$. When N is big and $\frac{1}{2}N^2 \gg 3N$, the savings become significant.

For $N = 2^q$, the lemma may be applied recursively for further savings:

"Fast" Fourier Transform, or FFT, on $N = 2^q$ Points

```
fft( v, N ):
[0] If N>1, then do [1] to [9]
[1]    For k=0 to N/2-1, let ve[k] = v[2*k]
[2]    Compute fft(ve, N/2)
[3]    For k=0 to N/2-1, let vo[k] = v[2*k+1]
[4]    Compute fft(vo, N/2)
[5]    For m=0 to N/2-1, do [6] to [9]
[6]       Let w.Re = cos(2*PI*m/N)
[7]       Let w.Im = -sin(2*PI*m/N)
[8]       Let v[m] = ve[m] + w*vo[m]
[9]       Let v[m+N/2] = ve[m] - w*vo[m]
```

The input/output array `v[]`, the temporary arrays `ve[]` and `vo[]`, and the scalar `w` are all complex-valued. We denote the real and imaginary parts of `w`, for example, by `w.Re` and `w.Im`, respectively. The operators `=`, `*` and `+` signify complex assignment, complex addition, and complex multiplication, respectively.

The total cost of FFT for $N = 2^q$ is $O(Nq) = O(N \log_2 N)$ as $N \to \infty$. This has low complexity compared with the $O(N^2)$ cost of applying the dense matrix F_N directly to a vector.

The inverse discrete Fourier transform has virtually the same matrix as the discrete Fourier transform, only with $+i$ rather than $-i$ in the exponential function: $F^{-1} = \overline{F}$. But F is also a symmetric matrix, so another way of saying this is that $F^{-1} = F^*$, or that the Fourier transform matrix is unitary. Thus, inverse FFT is obtained from FFT by modifying just the one line defining the imaginary part of `w`:

```
[7']  Let w.Im = sin(2*PI*m/N)
```

FFT plays an enormous role in numerical analysis and signal processing, and as a consequence there exist many specialized and highly-engineered versions. Some of the more exotic variants are described by Walker in his book *Fast Fourier Transforms*. Our version is a relatively simple FFT implementation, popularized in a 1965 article by Cooley and Tukey.

3.2.2 Discrete Hartley transform

The *discrete Hartley transform*, or *DHT*, is a more symmetric and purely real-valued alternative to the discrete Fourier transformation. Its matrix $H_N : \mathbf{R}^N \to \mathbf{R}^N$ in the N-point case is given by the following formula:

$$H_N(m,n) = \operatorname{cas}\left(\frac{2\pi mn}{N}\right) \quad \stackrel{\text{def}}{=} \quad \cos\left(\frac{2\pi mn}{N}\right) + \sin\left(\frac{2\pi mn}{N}\right). \tag{3.54}$$

Then DHT is the linear transformation $v \mapsto \frac{1}{\sqrt{N}} H_N v$. Since $\operatorname{cas}\theta = \sqrt{2}\cos(\theta - \frac{\pi}{4}) = \sqrt{2}\sin(\theta + \frac{\pi}{4})$, the Hartley basis function $\operatorname{cas}(2\pi mn/N)/\sqrt{N}$ can be written as a just

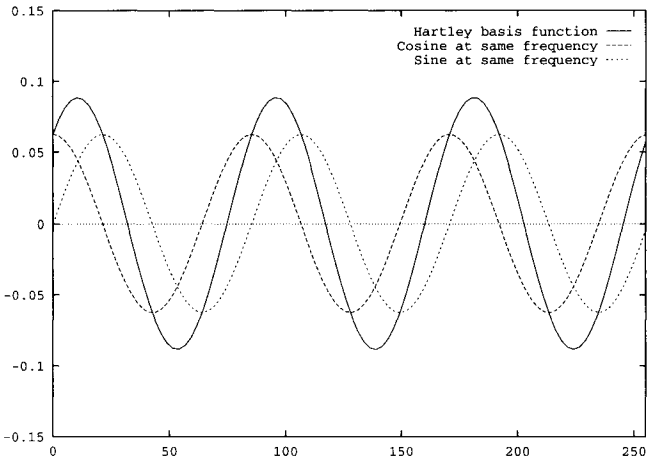


Figure 3.9: Hartley basis function $\frac{1}{16}\left[\cos\frac{2\pi 3n}{256} + \sin\frac{2\pi 3n}{256}\right]$.

one shifted sine or cosine, with amplitude $\sqrt{2/N}$. Figure 3.9 compares one example of such a basis function with sine and cosine functions of the same frequency m.

DHT can be obtained by slightly modifying the FFT implementation, or else by composing it with a small number of additional low complexity transformations. It can thus be factored in virtually the same way as DFT, using slightly different sparse factors. We begin with the following observation:

$$\operatorname{cas}\left(\frac{2\pi mn}{N}\right) = \Re\left[e^{\frac{2\pi imn}{N}} + ie^{-\frac{2\pi imn}{N}}\right] = \Im\left[e^{\frac{2\pi imn}{N}} + ie^{-\frac{2\pi imn}{N}}\right]. \tag{3.55}$$

For purely real $v \in \mathbf{R}^N$, we may therefore compute

$$H_N v(m) = \Re\left[F_N v(-m) + iF_N v(m)\right] = \Im\left[F_N v(-m) + iF_N v(m)\right]. \tag{3.56}$$

Thus a "fast" DHT can be had using FFT on these pieces. But the DHT of a real vector $f \in \mathbf{R}^N$ can also be performed purely in real arithmetic, using its own version of Lemma 3.13:

Lemma 3.14 *If N is an even positive integer, then the N-point discrete Hartley transform H_N splits into three parts as follows: for $m = 0, \ldots, N-1$,*

$$H_N v(m) = H_{\frac{N}{2}} v^e(m) + \cos(2\pi m/N)\, H_{\frac{N}{2}} v^o(m) + \sin(2\pi m/N)\, H_{\frac{N}{2}} v^o(-m).$$

Here $v^e(n) \stackrel{\text{def}}{=} v(2n)$ and $v^o(n) \stackrel{\text{def}}{=} v(2n+1)$ for $0 \le n < \frac{N}{2}$, and the $\frac{N}{2}$-point discrete Hartley transforms $H_{\frac{N}{2}}$ are considered $\frac{N}{2}$-periodic.

Proof: This is left as an exercise. □

This lemma may be applied recursively, as in the DFT case. To simplify the implementation, we use $\frac{N}{2}$-periodicity and the following identities:

$$\cos\left(2\pi\frac{m+\frac{N}{2}}{N}\right) = -\cos(2\pi\frac{m}{N}); \quad \sin\left(2\pi\frac{m+\frac{N}{2}}{N}\right) = -\sin(2\pi\frac{m}{N}).$$

"Fast" Hartley Transform, or FHT, on $N = 2^q$ Points

```
fht( v, N ):
[0]  If N>1, then do [1] to [11]
[1]     For k=0 to N/2-1, let ve[k] = v[2*k]
[2]     Compute fht(ve, N/2)
[3]     For k=0 to N/2-1, let vo[k] = v[2*k+1]
[4]     Compute fht(vo, N/2)
[5]     For m=1 to N/2-1, do [6] to [9]
[6]        Let c = cos(2*PI*m/N)
[7]        Let s = sin(2*PI*m/N)
[8]        Let v[m] = ve[m] + c*vo[m] + s*vo[N/2-m]
[9]        Let v[m+N/2] = ve[m] - c*vo[m] - s*vo[N/2-m]
[10]    Let v[0] = ve[0] + vo[0]
[11]    Let v[N/2] = ve[N/2] - vo[N/2]
```

3.2.3 Discrete sine and cosine transforms

There are other real-valued transforms similar to DFT and DHT. Eight of these, named as in Rao and Yip's *Discrete Cosine Transform*, are given in Table 3.1. In each tabulated matrix, the indices start at zero. The numbers $b(k)$ in the formulas are weights needed for orthogonality:

$$b(k) = \begin{cases} 0, & \text{if } k < 0 \text{ or } k > N; \\ 1/\sqrt{2}, & \text{if } k = 0 \text{ or } k = N; \\ 1, & \text{if } 0 < k < N. \end{cases} \tag{3.57}$$

DCT-IV, which needs no weights, is especially useful, so several of its properties will be proved.

Theorem 3.15 *The $N \times N$ matrix C_N^{IV} is symmetric and orthogonal.*

Proof: Symmetry is evident since the (m, n) entry is unchanged when m and n are swapped.

To show orthogonality, it remains to establish that for any $0 \le m, n \le N-1$,

$$\sum_{k=0}^{N-1} \left(\cos\frac{\pi(m+\frac{1}{2})(k+\frac{1}{2})}{N}\right)\left(\cos\frac{\pi(k+\frac{1}{2})(n+\frac{1}{2})}{N}\right) = \begin{cases} \frac{N}{2}, & \text{if } m = n, \\ 0, & \text{if } m \neq n. \end{cases} \tag{3.58}$$

Name	Rank	Matrix coefficient $A(m,n)$	Index range
DCT-I	$C_{N+1}^{I} : \mathbf{R}^{N+1} \to \mathbf{R}^{N+1}$	$b(m)b(n)\cos\frac{\pi mn}{N}$	$0, 1, \ldots, N-1, N$
DCT-II	$C_{N}^{II} : \mathbf{R}^{N} \to \mathbf{R}^{N}$	$b(m)\cos\frac{\pi m(n+\frac{1}{2})}{N}$	$0, 1, \ldots, N-1$
DCT-III	$C_{N}^{III} : \mathbf{R}^{N} \to \mathbf{R}^{N}$	$b(n)\cos\frac{\pi(m+\frac{1}{2})n}{N}$	$0, 1, \ldots, N-1$
DCT-IV	$C_{N}^{IV} : \mathbf{R}^{N} \to \mathbf{R}^{N}$	$\cos\frac{\pi(m+\frac{1}{2})(n+\frac{1}{2})}{N}$	$0, 1, \ldots, N-1$
DST-I	$S_{N-1}^{I} : \mathbf{R}^{N-1} \to \mathbf{R}^{N-1}$	$\sin\frac{\pi mn}{N}$	$1, \ldots, N-1$
DST-II	$S_{N}^{II} : \mathbf{R}^{N} \to \mathbf{R}^{N}$	$b(m+1)\sin\frac{\pi(m+1)(n+\frac{1}{2})}{N}$	$0, 1, \ldots, N-1$
DST-III	$S_{N}^{III} : \mathbf{R}^{N} \to \mathbf{R}^{N}$	$b(n+1)\sin\frac{\pi(m+\frac{1}{2})(n+1)}{N}$	$0, 1, \ldots, N-1$
DST-IV	$S_{N}^{IV} : \mathbf{R}^{N} \to \mathbf{R}^{N}$	$\sin\frac{\pi(m+\frac{1}{2})(n+\frac{1}{2})}{N}$	$0, 1, \ldots, N-1$

Table 3.1: Various discrete trigonometric transforms and their matrices.

But $2\cos A\cos B = \cos(A-B) + \cos(A+B)$, so the sum can be rewritten in two parts:

$$\frac{1}{2}\sum_{k=0}^{N-1}\cos\frac{\pi(k+\frac{1}{2})(m-n)}{N} + \frac{1}{2}\sum_{k=0}^{N-1}\cos\frac{\pi(m+n+1)(k+\frac{1}{2})}{N} \stackrel{\text{def}}{=} I + II.$$

Since $\exp(i\theta) = \cos\theta + i\sin\theta$, each of these can be rewritten as the real part of a geometric series:

$$\begin{aligned} I &= \frac{1}{2}\Re\sum_{k=0}^{N-1}\exp\left(\frac{i\pi(m-n)(k+\frac{1}{2})}{N}\right); \\ II &= \frac{1}{2}\Re\sum_{k=0}^{N-1}\exp\left(\frac{i\pi(m+n+1)(k+\frac{1}{2})}{N}\right). \end{aligned}$$

Suppose first that $n \neq m$. Then $-N+1 \leq m-n \leq N-1$, so $(m-n)/N$ is not an even integer. Likewise, $1 \leq m+n+1 \leq 2N-1$, so $(m+n+1)/N$ is not an even integer. The geometric summation formula can therefore be used:

$$\begin{aligned} I &= \Re\exp\left(\frac{i\pi(m-n)}{2N}\right)\frac{1-\exp(i\pi(m-n))}{1-\exp(i\pi(m-n)/N)} \\ &= \Re\frac{i(1\pm 1)}{2\sin(\pi(m-n)/(2N))} = 0, \end{aligned}$$

and

$$\begin{aligned} II &= \Re \exp\left(\frac{i\pi(m+n+1)}{2N}\right) \frac{1-\exp\left(i\pi(m+n+1)\right)}{1-\exp\left(i\pi(m+n+1)/N\right)} \\ &= \Re \frac{i(1\pm 1)}{2\sin\left(\pi(m+n+1)/(2N)\right)} \quad = 0, \end{aligned}$$

because $2i\sin\theta = \exp(i\theta) - \exp(-i\theta)$, and each expression is the real part of a purely imaginary number.

On the other hand, if $m = n$, then Equation 3.58 holds if

$$\sum_{k=0}^{N-1} \cos^2\left(\frac{\pi(n+\frac{1}{2})(k+\frac{1}{2})}{N}\right) = \frac{N}{2}.$$

But for any integer n and all real θ, $\cos\theta = (-1)^n \sin\left(\pi(n+\frac{1}{2}) - \theta\right)$, so

$$\begin{aligned} \sum_{k=0}^{N-1} \cos^2\left(\frac{\pi(n+\frac{1}{2})(k+\frac{1}{2})}{N}\right) &= \sum_{k=0}^{N-1} \sin^2\left(\frac{\pi(n+\frac{1}{2})(N-1-k+\frac{1}{2})}{N}\right) \\ &= \sum_{k=0}^{N-1} \sin^2\left(\frac{\pi(n+\frac{1}{2})(k+\frac{1}{2})}{N}\right), \end{aligned} \tag{3.59}$$

after the substitution $k \leftarrow N-1-k$. Finally, since $\sin^2\theta + \cos^2\theta = 1$ for all θ,

$$\sum_{k=0}^{N-1} \cos^2\left(\frac{\pi(n+\frac{1}{2})(k+\frac{1}{2})}{N}\right) = \frac{1}{2}\sum_{k=0}^{N-1} 1 = \frac{N}{2},$$

completing the proof. □

Noting that $2\sin A \sin B = \cos(A-B) - \cos(A+B)$ and $\sin^2\theta = \cos^2[\pi(n+\frac{1}{2}) - \theta]$, it is easy to see how the previous proof can be modified to give the following very similar result:

Corollary 3.16 *The $N \times N$ matrix S_N^{IV} is symmetric and orthogonal.* □

3.3 Exercises

1. Find the 1-periodization of the function $f(x) = e^{-|x|}$.

2. For real $\epsilon > 0$ and α, define the *dilation operator* δ_ϵ and the *translation operator* τ_α, which act on functions $f = f(t)$ of one real variable as follows:

$$\tau_\alpha u(t) \stackrel{\text{def}}{=} u(t-\alpha); \qquad \delta_\epsilon u(t) \stackrel{\text{def}}{=} \epsilon^{-1/2} u(t/\epsilon). \tag{3.60}$$

   a. Show that these are linear transformations with inverses $\tau_\alpha^{-1} = \tau_{-\alpha}$ and $\delta_\epsilon^{-1} = \delta_{1/\epsilon}$.

b. Compute the compositions $\tau_\alpha[\delta_\epsilon w](t)$ and $\delta_\epsilon[\tau_\alpha w](t)$ on a function $w = w(t)$.

c. Show that the fraying operator $F(r, \alpha, \epsilon)$ over reach interval $[\alpha - \epsilon, \alpha + \epsilon]$, as defined in Equation 3.20, may be written as $F(r, \alpha, \epsilon) = \tau_\alpha \delta_\epsilon F \delta_\epsilon^{-1} \tau_\alpha^{-1}$.

d. Derive the formula for splicing over the reach interval $[\alpha - \epsilon, \alpha + \epsilon]$ by computing $S(r, \alpha, \epsilon) = \tau_\alpha \delta_\epsilon S \delta_\epsilon^{-1} \tau_\alpha^{-1}$. Deduce that $F(r, \alpha, \epsilon)$ and $S(r, \alpha, \epsilon)$ are inverses.

3. Suppose that $u = u(x)$ is T-periodic. Show that the smooth local periodization of u to the interval $[0, T]$ equals u.

4. Show that the following operators can also be used for fraying and splicing:

$$\tilde{F}u(t) = \begin{cases} \bar{r}(t)u(t) - \bar{r}(-t)u(-t), & \text{if } t > 0, \\ r(-t)u(t) + r(t)u(-t), & \text{if } t < 0, \\ u(0), & \text{if } t = 0; \end{cases} \tag{3.61}$$

$$\tilde{S}u(t) = \begin{cases} r(t)u(t) + \bar{r}(-t)u(-t), & \text{if } t > 0, \\ \bar{r}(-t)u(t) - r(t)u(-t), & \text{if } t < 0, \\ u(0), & \text{if } t = 0. \end{cases} \tag{3.62}$$

That is, show that

(i) $\tilde{S}u(t) = u(t)$ and $\tilde{F}u(t) = u(t)$ if $|t| > 1$.

(ii) $\tilde{S}$ and $\tilde{F}$ are linear transformations of functions on $\mathbf{R}$.

(iii) $\tilde{S}$ and $\tilde{F}$ are inverses.

(iv) If u and r belong to $C^d(\mathbf{R})$, then for $0 \leq n \leq d$,

$$[\tilde{F}u]^{(n)}(0-) = 0 \quad \text{for odd } n; \qquad [\tilde{F}u]^{(n)}(0+) = 0 \quad \text{for even } n.$$

(v) If r belongs to $C^d(\mathbf{R})$ and u belongs to $C^d(\mathbf{R} \setminus \{0\})$ and has one-sided limits $u^{(n)}(0\pm)$ for $0 \leq n \leq d$ which satisfy

$$u^{(n)}(0-) = 0 \quad \text{for odd } n; \qquad u^{(n)}(0+) = 0 \quad \text{for even } n,$$

then defining $\tilde{S}u(0) = Su(0+)$ yields $\tilde{S}u \in C^d(\mathbf{R})$.

What happens if we substitute $r \leftarrow \bar{r}$? What if $r = \bar{r}$?

5. Prove, using induction and Leibniz' rule, that the function $r_n(t)$ defined below has $2^n - 1$ vanishing derivatives at $t = +1$ and $t = -1$, so that r_n has $2^n - 1$ continuous derivatives:

$$r_0(t) = \begin{cases} 0, & \text{if } t \leq -1, \\ \sin\left[\frac{\pi}{4}(1+t)\right], & \text{if } -1 < t < 1, \\ 1, & \text{if } t \geq 1. \end{cases};$$

$$r_n(t) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } t \leq -1, \\ r_{n-1}(\sin \frac{\pi}{2} t), & \text{if } -1 < t < 1, \\ 1, & \text{if } t \geq 1. \end{cases}$$

6. Show that the set of functions $\{1, \sqrt{2}\cos 2\pi nt, \sqrt{2}\sin 2\pi nt : n = 1, 2, \ldots\}$ is orthonormal with respect to the (Hermitean) inner product

$$\langle f, g\rangle \stackrel{\text{def}}{=} \int_0^1 \bar{f}(t)g(t)\,dt.$$

That is, show that

$$\left\langle \sqrt{2}\cos 2\pi nt, \sqrt{2}\sin 2\pi mt \right\rangle = 0, \quad \text{all } n, m \in \mathbf{Z}^+; \tag{3.63}$$

$$\left\langle \sqrt{2}\cos 2\pi nt, \sqrt{2}\cos 2\pi mt \right\rangle = 0, \quad n \neq m \in \mathbf{Z}^+ \cup \{0\}; \tag{3.64}$$

$$\left\langle \sqrt{2}\sin 2\pi nt, \sqrt{2}\sin 2\pi mt \right\rangle = 0, \quad n \neq m \in \mathbf{Z}^+; \tag{3.65}$$

$$\|1\| = \|\sqrt{2}\sin 2\pi nt\| = \|\sqrt{2}\cos 2\pi nt\| = 1. \tag{3.66}$$

7. Compute the sine-cosine Fourier series of the 1-periodic function $f(x) = \cos^2(2\pi x)$. (Hint: use a trigonometric identity.)

8. Compute the complex exponential Fourier series of the 1-periodic function $\sin(2\pi kt - d)$, where d is a constant real number.

9. Show that if $|c(n)| < 1/|n|^3$ for all integers $n \neq 0$, then the 1-periodic function $f = f(t)$ which is the inverse Fourier transform of the sequence $\{c(n)\}$ must have a continuous derivative $f'(t)$ at every $t \in [0, 1]$.

10. Show that the functions $\phi_k(t) = \mathbf{1}(Nt - k)$, $k = 0, 1, \ldots, N-1$ are an orthogonal collection with respect to the inner product

$$\langle f, g\rangle \stackrel{\text{def}}{=} \int_0^1 \overline{f(t)}g(t)\,dt.$$

Here $\mathbf{1} = \mathbf{1}_{[0,1)}$ is the indicator function of the interval $[0, 1)$. How can this collection be made orthonormal? What is the linear span of $\{\phi_k : 0 \leq k < N\}$?

11. Suppose that ϕ has Fourier integral transform $\mathcal{F}\phi$. Let $\phi_k(x) \stackrel{\text{def}}{=} \phi(x - k)$. Show that $\mathcal{F}\phi_k(\xi) = e^{-2\pi ik\xi}\mathcal{F}\phi(\xi)$.

12. Suppose that ϕ has Fourier integral transform $\mathcal{F}\phi$. Let $\phi_a(x) \stackrel{\text{def}}{=} \phi(x/a)$, for $a > 0$. Show that $\mathcal{F}\phi_a(\xi) = a\mathcal{F}\phi(a\xi)$.

13. Compute the inverse Fourier integral transform of the function

$$\psi(\xi) = \begin{cases} 1, & \text{if } -1 \leq \xi < -\frac{1}{2} \text{ or } \frac{1}{2} < \xi \leq 1; \\ 0, & \text{otherwise.} \end{cases}$$

(Hint: notice that $\psi(\xi) = \mathbf{1}_I(\xi/2) - \mathbf{1}_I(\xi)$, where $I = [-\frac{1}{2}, \frac{1}{2}]$.)

14. Compute the Fourier integral transform of the hat function

$$h(x) = \begin{cases} 1 - |x|, & \text{if } -1 \leq x \leq 1; \\ 0, & \text{otherwise.} \end{cases}$$

15. Show that the vectors $\omega_n \in \mathbf{C}^N$, $n = 0, 1, \ldots, N-1$ defined by $\omega_n(k) = \frac{1}{\sqrt{N}} \exp(2\pi ink/N)$ form an orthonormal basis with respect to the inner product

$$\langle f, g \rangle \stackrel{\text{def}}{=} \sum_{k=0}^{N-1} \overline{f(k)}\, g(k).$$

16. Write out explicitly the matrices for the 2×2 and 4×4 discrete Fourier and Hartley transforms (F_2, F_4, H_2 and H_4).

17. What is the matrix of the square of $N \times N$ *DFT*? Give a formula for every positive integer N.

18. What is the matrix of the fourth power of $N \times N$ *DFT*? Give a formula for every positive integer N.

19. Write out explicitly the matrices C_2^{IV} and C_4^{IV} used for the 2×2 and 4×4 DCT-IV, respectively.

20. Prove Lemma 3.14.

21. Prove Corollary 3.16.

3.4 Further Reading

- Tom M. Apostol. *Mathematical Analysis.* Addison–Wesley, Reading, Massachusetts, second edition, 1974. ISBN 0-201-00288-4.
- James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.
- Jean Baptiste Joseph Fourier. *Théorie Analytique de la Chaleur.* F. Didot, Paris, 1822.
- K. R. Rao and P. Yip. *Discrete Cosine Transform.* Academic Press, New York, 1990. ISBN 0-12-580203-X.
- Elias M. Stein and Guido Weiss. *Introduction to Fourier Analysis on Euclidean Spaces.* Number 32 in the Princeton Mathematical Series. Princeton University Press, Princeton, New Jersey, 1971. ISBN 0-691-08078-X.

- James S. Walker. *Fast Fourier Transforms.* CRC Press, Boca Raton, Florida, 1992. ISBN 0-8493-7154-6.

- Gregory K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34:30–44, 1991. Also available from URL `ftp://ftp.uu.net/graphics/jpeg/wallace.ps.gz`.

Chapter 4

Sampling and Estimation

Multimedia signal processing begins with the taking of physical measurements to acquire sounds or images. To determine what measurements are needed, we use a mathematical model of the physical process. For example, speech recording begins with a model of the sound vibrations that a human being can produce. These result in slight fluctuations of the air pressure that can be measured by a microphone, producing a real-valued function of one real "time" variable. Similarly, taking a picture begins with a model of the light arriving at the camera. The amount that arrives during the acquisition period, or when the shutter is open, is a nonnegative real-valued function of two "space" variables. Finally, video can be modeled by a nonnegative function of one time plus two space variables.

There is no general method for representing arbitrary functions of one, two, or three real variables in a computer. That would require an infinite amount of information even in some simple cases. However, the functions produced or consumed by humans as sounds, pictures, or videos are not arbitrary. They are very smooth functions since humans can neither move too fast nor absorb very rapidly changing signals. Hence *sampling* the functions often enough, or at enough places, will give a reliable approximation. The smoothness, or slow-change assumption, implies that no large fluctuations occur between sample points. We will study methods for recovering a function from its samples, given this assumption.

Looking deeper, we see that the physical measurement of a sample contains numerous uncertainties. We will obtain estimates of the errors caused by our fundamental ignorance by modeling our measurements as random variables and using tools from probability theory.

The uncertainty and error in physical measurement is actually a blessing in disguise. On the one hand, the data in our computer can never be a perfect representation of reality, and will always be corrupted with *noise*. On the other hand, our processing algorithms are freed from the tyranny of exactness. We can use fast approximate computation methods in many cases, if we keep the calculation errors small compared to the original measurement errors.

4.1 Approximation and Sampling

We will restrict ourselves to the problem of approximating one real or complex-valued function $f = f(x)$ of one real variable x, given its values at discrete points. For the moment, we will assume that the values and the sample points are known exactly.

So, let $\{x_k : 0 \leq k \leq n\}$ be a set of $n+1$ distinct real numbers. Then for any real or complex numbers $\{y_k : 0 \leq k \leq n\}$, the *sampling set* $S = \{(x_k, y_k) : 0 \leq k \leq n\}$ is a list of $n+1$ distinct points. For example, we may take S to be a table of *sample points* $y_k = f(x_k)$ for f. If $x_k = x_0 + kh$ for some $h > 0$ and all $k = 0, 1, \ldots, n$, then S is called a *regular* or *equally-spaced* sampling of f with *sampling interval* h.

Given S and some assumptions about f, it is often desirable to compute $f(x)$ at points $x \notin \{x_k\}$. This process $S \mapsto f$ is called *interpolation*, and has a long history with many applications. For example, one of the earliest kinds of compressed data was tables of sines and cosines recorded to high precision at just a few angles. With high-order *polynomial interpolation*, any other desired value could be computed with equally high precision in just a few operations. A more recent application is *resampling* a function at other points, to accommodate new display devices. This can often be done speedily with low-order *piecewise approximation*. Finally, some operations such as differentiating or integrating f reduce to matrix multiplication, if we know that f belongs to a *sampling space*, and has an expansion in known basis functions.

4.1.1 Polynomial interpolation

Any single value of a *polynomial* $p = p(x)$ can be computed in finitely many steps from its formula:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n. \tag{4.1}$$

The highest power of x, here n, is called the *degree* of p. Clearly $n \geq 0$. The $n+1$ numbers $\{a_0, a_1, \ldots, a_n\}$ are called the *coefficients* of p.

Given the coefficients $\{a_0, a_1, \ldots, a_n\}$ and a fixed x, it costs $O(n)$ floating-point operations to compute $p(x)$ by *Horner's method*, which works by grouping operations as follows:

$$p(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-2} + x(a_{n-1} + xa_n)) \cdots)). \tag{4.2}$$

Below is one implementation:

Polynomial Evaluation by Horner's Method

```
horner( x, a[], n ):
[0] Let y = a[n]
[1] For k=n-1 down to 0, do [2]
[2]    Replace y = a[k] + x*y
[3] Return y
```

Step 2 consists of one multiplication and one addition. It is executed n times, making the cost $O(n)$ operations per evaluation.

Any number z for which $p(z) = 0$ is called a root of p, and we can write $p(x) = (x - z)q(x)$, where $q(x)$ is a polynomial of degree one less than p. This implies that if the degree is n, there can be at most n distinct roots. Another way of saying this is:

Lemma 4.1 *If the degree of a polynomial p is known to be n or less, but it is known to have more than n distinct roots, then all of p's coefficients must be zero and thus $p(x) = 0$ for all numbers x.*

Proof: If $z_0, z_1, z_2, \ldots, z_n$ are $n+1$ distinct roots of $p = p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1} + a_n x^n$, then the following matrix equation holds:

$$\begin{pmatrix} 1 & z_0 & z_0^2 & \cdots & z_0^n \\ 1 & z_1 & z_1^2 & \cdots & z_1^n \\ 1 & z_2 & z_2^2 & \cdots & z_2^n \\ \vdots & & \ddots & & \vdots \\ 1 & z_n & z_n^2 & \cdots & z_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{4.3}$$

Induction on n is next used to prove that the determinant of the matrix is

$$\prod_{0 \le i < j \le n} (z_j - z_i).$$

Since $z_i \neq z_j$ for $i \neq j$, the matrix is nonsingular, so the only solution to the linear system is $a_0 = a_1 = \cdots = a_n = 0$. □

The quadratic formula gives all the roots of $p(x) = a_0 + a_1 x + a_2 x^2$ when the degree is two:

$$z_1 = \frac{-a_1 + \sqrt{a_1^2 - 4a_0 a_2}}{2a_2}; \qquad z_2 = \frac{-a_1 - \sqrt{a_1^2 - 4a_0 a_2}}{2a_2}.$$

This formula uses only $+, -, \times, \div$, and $\sqrt{\ }$. There are also the more complicated Cardano formulas for roots of third and fourth degree polynomials. which use higher-order radicals as well, but it is a deep fact discovered in the 1820's that no formula using just the four basic operations plus radicals of any order can represent the roots in all cases when the degree is five or more.

Lagrange interpolation

If $f = f(x)$ is a polynomial of degree n, then its coefficients $\{a_0, \ldots, a_n\}$ are uniquely determined by the values $f(x)$ at $n+1$ distinct points $\{x_0, \ldots, x_n\}$. The algorithm for finding the coefficients is called *Lagrange interpolation*, and is based on the following identity for polynomials of degree n:

$$\Lambda_n(x) = \sum_{k=0}^{n} y_k \lambda_k(x), \tag{4.4}$$

where for every $k = 0, 1, \ldots, n$,

- $y_k = f(x_k)$ are the fixed evaluations of f at $x_0, x_1, \ldots, x_n$, and
- $\lambda_k(x) = \prod_{j \neq k} \frac{x - x_j}{x_k - x_j}$ is a polynomial of degree n.

Theorem 4.2 *If $f = f(x)$ is a polynomial of degree n or less, and Λ_n is defined by Equation 4.4, then $f(x) = \Lambda_n(x)$ for every x.*

Proof: Notice that $\lambda_k(x_k) = 1$ while $\lambda_k(x_j) = 0$ for $j \neq k$, so that $\Lambda_n(x_k) = f(x_k)$ for $k = 0, 1, \ldots, n$. Thus the polynomial $f(x) - \Lambda_n(x)$, which has degree n, has $n + 1$ distinct roots $x_0, x_1, \ldots, x_n$ and must be identically zero. □

The algorithm to evaluate a polynomial of degree n at an arbitrary point x, given its values at $n+1$ distinct points, requires $2n+2$ inputs and $O(n^2)$ arithmetic operations:

Lagrange Polynomial Evaluation

```
lagrange( x, x[], y[], n ):
[0] Let y = 0
[1] For k=0 to n, do [2] to [5]
[2]    Let lambda = y[k]
[3]    For j=0 to n, do [4]
[4]       If j != k, multiply lambda *= (x-x[j])/(x[k]-x[j])
[5]    Accumulate y += lambda
[6] Return y
```

Newton's form prepares the Lagrange polynomial for evaluation by a generalization of Horner's method. Instead of superposing the values of $n + 1$ elementary polynomials $\lambda_k(x)$, $k = 0, \ldots, n$, each of degree n and determined by $\{x_0, \ldots, x_n\}$, we precompute $n + 1$ coefficients $c_0, c_1, c_2, \ldots, c_n$ such that

$$\Lambda_n(x) = c_0 + (x - x_0)[c_1 + (x - x_1)[c_2 + \cdots + (x - x_{n-1})c_n] \cdots]. \tag{4.5}$$

Expanded, this form bears some resemblance to the Taylor series for Λ_n:

$$\Lambda_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0) \cdots (x - x_{n-1}).$$

Instead of j^{th} derivatives, the coefficients are given by $c_j = D^j f(x_j)$, where $D^j f$ is the j^{th} *divided difference*, defined recursively as follows:

$$\begin{aligned} D^0 f(x_k) &\stackrel{\text{def}}{=} f(x_k), \qquad k = 0, 1, \ldots, n; \\ D^j f(x_k) &= \frac{D^{j-1} f(x_k) - D^{j-1} f(x_{k-1})}{x_k - x_{k-j}}, \qquad 0 < j \leq k \leq n. \end{aligned}$$

In the special case of equispaced sampling, with $x_k = x_0 + kh$ for fixed $h > 0$, the Newton form mimics the Taylor series even more closely:

$$\Lambda_n(x) = \sum_{j=0}^{n} c_j \prod_{i=0}^{j-1} (x - x_i) = \sum_{j=0}^{n} \frac{\Delta^j f(x_j)}{j! h^j} \prod_{i=0}^{j-1} (x - x_i), \tag{4.6}$$

where the coefficients $c_j = \frac{\Delta^j f(x_j)}{j!h^j}$ are computed from ordinary undivided j^{th} differences $\Delta^j f$ defined recursively as follows:

$$\begin{aligned} \Delta^0 f(x_k) &\stackrel{\text{def}}{=} f(x_k), \qquad k = 0, 1, \ldots, n; \\ \Delta^j f(x_k) &= \Delta^{j-1} f(x_k) - \Delta^{j-1} f(x_{k-1}), \qquad 0 < j \leq k \leq n. \end{aligned}$$

Newton's form allows easier updates of the interpolating polynomial in response to additional points. For example, if one new point $(x_{n+1}, f(x_{n+1}))$ is added to the interpolation set $\{(x_k, f(x_k)) : k = 0, 1, \ldots, n\}$, then to get the new Lagrange polynomial we simply add one more term to the old Newton form:

$$\Lambda_{n+1}(x) = \Lambda_n(x) + c_{n+1} \prod_{k=0}^{n} (x - x_k), \tag{4.7}$$

where $c_{n+1} = \frac{\Delta^{n+1} f(x_{n+1})}{(n+1)!h^{n+1}}$ in the equispaced case, or $c_{n+1} = D^{n+1} f(x_{n+1})$ in general.

To compute $c_0, \ldots, c_n$ from the interpolation set $\{(x_k, f(x_k)) : k = 0, \ldots, n\}$ costs $O(n^2)$ operations, using the following algorithm:

Divided Differences for Newton's Form

```
divdiff( c[], x[], y[], n ):
[0] For k=0 to n, let c[k] = y[k]
[1] For j=1 to n, do [2] to [3]
[2]    For k=n down to j, do [3]
[3]       Replace c[k] = (c[k]-c[k-1])/(x[k]-x[k-j])
```

Given $\{c_k\}$, subsequent evaluations of Λ_n cost $O(n)$ operations each:

Newton's Evaluation of the Lagrange Polynomial

```
newtonpoly(x, x[], c[], n ):
[0]  Let y = c[n]
[1]  For k=n-1 down to 0, replace y = c[k] + (x-x[k])*y
[2]  Return y
```

Chebyshev polynomials

If we can choose the sampling points $\{x_k\}$, then we can select a good representation of the Lagrange polynomial due to Chebyshev. First note that any function $f = f(x)$ defined for $x \in [a, b]$ determines an equivalent function $g(y) = f \circ p(y)$ defined on $y \in [-1, 1]$, where $p(y) \stackrel{\text{def}}{=} \frac{b-a}{2}y + \frac{b+a}{2}$ maps $[-1, 1]$ to $[a, b]$. We also have have $f(x) = g \circ q(x)$, where $q(x) \stackrel{\text{def}}{=} \frac{2}{b-a}x - \frac{b+a}{b-a}$ maps $[a, b]$ to $[-1, 1]$. Since both p and q are of the form $Ax + B$ with $A \neq 0$, the functions f and g have equal smoothness and integrability. Also, f is a polynomial of degree n if and only if g is a polynomial

of degree n. Hence, we may assume without loss of generality that our functions are defined on $[-1, 1]$.

For integer $n \geq 0$, define the n^{th} *Chebyshev polynomial*, T_n, as follows:

$$T_n(x) \stackrel{\text{def}}{=} \cos(n \arccos x), \qquad x \in [-1, 1]. \tag{4.8}$$

That these are actually polynomials in x requires proof:

Lemma 4.3 *$T_0(x) = 1$, $T_1(x) = x$, and for all $n > 1$, T_n is a polynomial of degree n, with leading coefficient 2^{n-1}, satisfying the recursion relation*

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x).$$

Proof: First verify $T_0(x) = \cos 0 = 1$ and $T_1(x) = \cos(\arccos x) = x$. Next, write $\theta = \arccos x$, so $\cos\theta = x$, and note that for $n \geq 2$,

$$2xT_{n-1}(x) - T_{n-2}(x) = 2\cos\theta\cos[(n-1)\theta] - \cos[(n-2)\theta].$$

Then evaluate the second term,

$$\cos[(n-2)\theta] = \cos[(n-1)\theta - \theta] = \cos\theta\cos[(n-1)\theta] + \sin\theta\sin[(n-1)\theta],$$

so

$$\begin{aligned} 2xT_{n-1}(x) - T_{n-2}(x) &= \cos\theta\cos[(n-1)\theta] - \sin\theta\sin[(n-1)\theta] \\ &= \cos[n\theta] \quad = T_n(x). \end{aligned}$$

This establishes the recursion relation.

Since T_0 and T_1 are polynomials, the recursion relation implies that T_n is also a polynomial for all $n > 1$. It remains to establish, by induction on n, that the highest order term of $T_n(x)$ is $2^{n-1}x^n$ (for all $n > 0$, in fact), but this is left as an exercise. □

Lemma 4.4 *The roots of T_n are the n distinct numbers $x_k = \cos\left[\frac{\pi(k+\frac{1}{2})}{n}\right]$, $k = 0, 1, \ldots, n-1$, all of which lie in the interval $[-1, 1]$.*

Proof: There are at most n distinct roots of T_n, since it has degree n, and the numbers $\{x_k\}$ are evidently distinct and lie in $[-1, 1]$. It remains to check: $T_n(x_k) = \cos(n \arccos x_k) = \cos\left[\pi(k + \frac{1}{2})\right] = 0$. □

Lemma 4.5 *Any polynomial $p = p(x)$ of degree N or less can be written as a linear combination of the Chebyshev polynomials $T_0, T_1, \ldots, T_N$.*

Proof: This is obviously true for $N = 0$ and $N = 1$. Suppose that the result is true for $N-1 \geq 1$, namely, suppose $1, x, \ldots, x^{N-1}$ are expressible as linear combinations from the set $T_0, T_1, \ldots, T_{N-1}$. Since $x^N - 2^{-(N-1)}T_N(x)$ is a polynomial of degree

at most $N-1$, we see that x^N can be written as a linear combination of T_N and $T_0, T_1, \ldots, T_{N-1}$. By induction, the result holds for all integers $N \geq 0$. □

In particular, the Lagrange polynomial of degree at most N interpolating $N+1$ points $\{(x_k, y_k) : k = 0, \ldots, N\}$ with $\{x_k\} \subset [-1, 1]$ can be written as a sum of Chebyshev polynomials:

$$\Lambda_N(x) = \sum_{n=0}^{N} c(n) T_n(x).$$

The expansion coefficients are easily computed from the sequence $\{y_k\}$ if we use *Chebyshev nodes*, the $N+1$ distinct roots of T_{N+1}:

$$x_k = \cos \frac{\pi(k + \frac{1}{2})}{N+1}, \qquad k = 0, 1, \ldots, N, \tag{4.9}$$

That is because Chebyshev polynomials $\{T_0, \ldots, T_N\}$ have a *discrete orthogonality property* with respect to the roots of T_{N+1}. Let P be the space of polynomials of degree N or less, with inner product

$$\langle p, q \rangle \stackrel{\text{def}}{=} \sum_{k=0}^{N} \overline{p(x_k)} q(x_k),$$

where $\{x_k\}$ is the set of Chebyshev nodes defined in Equation 4.9. This inner product is obviously linear and Hermitean symmetric. It is nondegenerate, since $\langle p, p \rangle = 0$ only if p has $N+1$ distinct roots, which forces $p = 0$ since the degree of $p \in P$ is N or less. For the polynomials T_n, T_m with $0 \leq n, m \leq N$, we get

$$\begin{aligned} \langle T_n, T_m \rangle &= \sum_{k=0}^{N} \cos(n \arccos x_k) \cos(m \arccos x_k) \\ &= \sum_{k=0}^{N} \left(\cos \frac{\pi(k + \frac{1}{2})n}{N+1} \right) \left(\cos \frac{\pi(k + \frac{1}{2})m}{N+1} \right), \end{aligned}$$

which is zero for $n \neq m$ because distinct rows of the DCT-II matrix in Table 3.1 are orthogonal.

Likewise, it is straightforward to show that $\langle T_0, T_0 \rangle = \sum_{k=0}^{N} 1 = N+1$, and

$$\langle T_n, T_n \rangle = \sum_{k=0}^{N} \cos^2 \left[\frac{\pi(k + \frac{1}{2})n}{N+1} \right] = \frac{N+1}{2}, \qquad n = 1, 2, \ldots, N. \tag{4.10}$$

Theorem 4.6 *For $k = 0, 1, \ldots, N$, suppose that $x_k = \cos \frac{\pi(k+\frac{1}{2})}{N+1}$ is a Chebyshev node and y_k is arbitrary. Then the Lagrange polynomial of degree N or less interpolating $\{(x_k, y_k) : k = 0, 1, \ldots, N\}$ is:*

$$\Lambda_N(x) = \sum_{m=0}^{N} c(m) T_m(x),$$

where $c(0) = \frac{1}{N+1}\sum_{n=0}^{N} y_n$, *and*

$$c(m) = \frac{2}{N+1}\sum_{n=0}^{N} y_n \cos\frac{\pi(n+\frac{1}{2})m}{N+1}, \qquad m = 1, 2, \ldots, N.$$

Proof: Every polynomial of degree N has the stated form, including the Lagrange polynomial $\Lambda_N(x)$. By definition,

$$\begin{aligned}\langle T_m, \Lambda_N\rangle &= \sum_{n=0}^{N} \overline{T_m(x_n)}\Lambda_N(x_n) \\ &= \sum_{n=0}^{N} \cos\left[\frac{\pi(n+\frac{1}{2})m}{N+1}\right]\Lambda_N(x_n) = \sum_{n=0}^{N} \cos\left[\frac{\pi(n+\frac{1}{2})m}{N+1}\right] y_n.\end{aligned}$$

This can also be evaluated using discrete orthogonality:

$$\langle T_m, \Lambda_N\rangle = \sum_{n=0}^{N} c(n)\,\langle T_m, T_n\rangle = c(n)\,\langle T_n, T_n\rangle\,.$$

Applying Equation 4.10 finishes the proof. □

Thus, given values $\{y_n\}$ at the $N+1$ Chebyshev nodes $\{x_n\}$, we can precompute the Chebyshev expansion coefficients $\{c(m)\}$ using the DCT-II transform:

$$c(m) = \sum_{n=0}^{N} C^{II}_{N+1}(m, n) y_n,$$

which is an investment of $O(N^2)$ operations. If $N+1$ is a power of 2, this may be done in $O(N \log N)$ operations with the fast factored DCT-II. Thereafter, evaluations cost only $O(N)$ operations each since the Chebyshev recursion relation permits efficient grouping:

Chebyshev's Evaluation of the Lagrange Polynomial

```
chebyshevpoly( x, c[], n ):
[0]  Let y = c[0]
[1]  If n>0, then do [2] to [7]
[2]     Replace y += x*c[1]
[3]     If n>1, then do [4] to [7]
[4]        Let T[0] = 1, let T[1] = x
[5]        For k=2 to n, do [6] to [7]
[6]           Let T[k] = 2*x*T[k-1] - T[k-2]
[7]           Replace y += c[k]*T[k]
[8]  Return y
```

Chebyshev polynomials also have a *continuous orthogonality property.* Consider the vector space $\mathbf{Lip}[-1, 1]$ of Lipschitz continuous functions on $[-1, 1]$, with inner

product $\langle u, v \rangle \stackrel{\text{def}}{=} \int_{-1}^{1} \frac{\overline{u(x)}v(x)}{\sqrt{1-x^2}}\,dx$. This contains all polynomials, including all the Chebyshev polynomials. By substituting $x \leftarrow \cos\theta$, so $d\theta = -(1-x^2)^{-1/2}\,dx$, we get

$$\langle T_n, T_m \rangle = \int_{-1}^{1} \frac{\cos(n \arccos x)\cos(m \arccos x)}{\sqrt{1-x^2}}\,dx = \int_{-\pi}^{\pi} \cos(n\theta)\cos(m\theta)\,d\theta.$$

The substitution $\theta \leftarrow 2\pi t - \pi$ converts this to

$$\langle T_n, T_m \rangle = \pm 2\pi \int_0^1 \cos(2\pi n t)\cos(2\pi m t)\,dt,$$

which is zero for $n \neq m$ by Equation 3.64.

4.1.2 Piecewise interpolation

High-order polynomials are sums of many terms that can have wildly different magnitudes. It is often better to approximate a function in pieces using low-order polynomials on each piece. Pieces will be short runs of sample points, so we now suppose that $x_0 < \cdots < x_N$, where N is fixed in advanced. The values $y_0, \ldots, y_N$ to be interpolated are arbitrary, and we denote the set of *interpolation points* by

$$S = \{(x_k, y_k) : k = 0, 1, \ldots, N\} \subset \mathbf{R}^2. \tag{4.11}$$

The *piecewise constant function* through S is defined by:

$$f(x) = y_k, \qquad \text{when } x \text{ is closest to } x_k. \tag{4.12}$$

Such f is also called a *step function*. The evaluation algorithm, which assumes that $n > 0$ and that $x_0 \leq x \leq x_n$, takes $O(n)$ operations:

Piecewise Constant Evaluation

```
pwconstant( x, x[], y[], N ):
[0] For k=0 to N-1, do [1]
[1]    If x < (x[k]+x[k+1])/2, then return y[k]
[2] Return y[N]
```

All of the work is in finding that k for which x is closest to x_k. If the points $\{x_k\}$ are *equispaced* in some interval $[a, b]$, so that $x_k = a + kh$ for $h = (b-a)/N$, then $k = \lfloor \frac{1}{2} + (x-a)/h \rfloor$ and each evaluation costs a fixed number $O(1)$ of operations, independent of N:

Equispaced Piecewise Constant Evaluation

```
epwconstant( x, y[], a, b, N ):
[0] Let k = floor( 0.5 + N*(x-a)/(b-a) )
[1] Return y[k]
```

The *piecewise linear function* through S is defined by:

$$f(x) = \frac{y_k(x - x_{k-1}) + y_{k-1}(x_k - x)}{x_k - x_{k-1}}, \qquad \text{when } x_{k-1} \le x \le x_k. \tag{4.13}$$

Notice that $f(x) = \Lambda_1(x)$ on the interval $[x_0, x_1]$, where Λ_1 is the degree-1 Lagrange polynomial given by Equation 4.4. On the general interval $[x_{k-1}, x_k]$, f is given by that same formula $f(x) = \Lambda_1(x)$, derived from the interpolation set $S_k = \{(x_{k-1}, y_{k-1}), (x_k, y_k)\}$.

The graph of f is obtained by connecting (x_{k-1}, y_{k-1}) to (x_k, y_k), for each $k = 1, 2, \ldots, N$, with a line segment. This is a simple way to obtain a graph from a table of at least two function values. The evaluation algorithm takes $O(N)$ operations, most of which are spent finding the right subinterval:

Piecewise Linear Evaluation

```
pwlinear( x, x[], y[], N ):
[0] If x < x[0], then return y[0]
[1] For k=1 to N, do [2]
[2]   If x < x[k], then return
        (y[k]*(x-x[k-1]) + y[k-1]*(x[k]-x)) / (x[k]-x[k-1])
[3] Return y[N]
```

This implementation is robust: it returns y_0 if $x \le x_0$ and returns y_N if $x \ge x_N$. The version for equispaced $\{x_n\}$ is left as an exercise.

To find a root of a function from its samples, we first locate two adjacent samples y_1, y_2 of opposite sign, then use the root x_0 of the linear function interpolating $\{(x_1, y_1), (x_2, y_2)\}$:

$$x_0 = \frac{x_2 y_1 - x_1 y_2}{y_1 - y_2}. \tag{4.14}$$

To find a local maximum or minimum of a smooth function f from its samples $y_k = f(x_k)$, we first locate three consecutive samples y_1, y_2, y_3 such that $y_2 - y_1$ and $y_3 - y_2$ are of opposite sign. Then the Mean Value Theorem implies that $f'(x)$ changes sign between x_1 and x_3, and the Intermediate Value Theorem implies that $f'(x_0) = 0$ for some $x_0 \in (x_1, x_3)$. We approximate this extremum x_0 with the vertex of the parabola $y = \Lambda_2(x)$ interpolating $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$. That quadratic polynomial may be written in efficient-to-evaluate Newton form as follows:

$$y = \Lambda_2(x) = y_1 + (x - x_1)\left(\frac{y_2 - y_1}{x_2 - x_1} + (x - x_2)\frac{\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1}}{x_3 - x_1}\right). \tag{4.15}$$

The extreme value is then approximated by $y_* \stackrel{\text{def}}{=} \Lambda_2(x_*)$, where x_* is the unique critical point satisfying $\Lambda_2'(x_*) = 0$. Solving for x_* gives

$$x_* = \frac{1}{2}\frac{x_1^2(y_3 - y_2) + x_2^2(y_1 - y_3) + x_3^2(y_2 - y_1)}{x_1(y_3 - y_2) + x_2(y_1 - y_3) + x_3(y_2 - y_1)}. \tag{4.16}$$

We may also combine the interpolation and root formulas to get y_* directly:

$$y_* = \frac{2\left[y_1^2\Delta x_{23}^4 + y_2^2\Delta x_{31}^4 + y_3^2\Delta x_{12}^4\right] - \left[y_1\Delta x_{23}^2 + y_2\Delta x_{31}^2 + y_3\Delta x_{12}^2\right]^2}{4\Delta x_{12}\Delta x_{23}\Delta x_{31}\left[y_1\Delta x_{23} + y_2\Delta x_{31} + y_3\Delta x_{12}\right]}. \quad (4.17)$$

Here $\Delta x_{ij} = x_i - x_j$ for $i, j \in \{1, 2, 3\}$.

One way to check such complicated expressions is by symmetry. It cannot matter how we label the points, so the formulas $x_*((x_1, y_1), (x_2, y_2), (x_3, y_3))$ and $y_*((x_1, y_1), (x_2, y_2), (x_3, y_3))$ must be *symmetric functions*, left unchanged by all permutations of the indices $(1, 2, 3)$.

Another check is based on rescaling: for any $s, t \neq 0$, changing x_1, x_2, x_3 to sx_1, sx_2, sx_3 must change x_* to sx_* leaving y_* fixed, and changing y_1, y_2, y_3 to ty_1, ty_2, ty_3 must change y_* to ty_* leaving x_* fixed. In other words, the units for x_* and y_* are the units of the x and y axes, respectively. Such *scale invariance* is expressed more precisely by the formulas

$$x_*(sx, ty) = s\,x_*(x, y); \qquad y_*(sx, ty) = t\,x_*(x, y).$$

It is a worthwhile exercise to confirm that Equations 4.16 and 4.17 are scale invariant symmetric functions.

4.1.3 Sampling spaces

A function $\phi = \phi(x)$ of one real variable is said to satisfy an *interpolation condition* if the following holds at the integers:

$$\phi(0) = 1; \qquad \phi(n) = 0, \quad \text{if } n = \pm 1, \pm 2, \ldots. \quad (4.18)$$

Such a function generates a *sampling space*, which consists of all the linear combinations of integer translates of ϕ, or $u(x) = \sum_n c(n)\phi(x-n)$, where $c = \{c(n) : n \in \mathbf{Z}\}$ is a sequence of scalars. But then the interpolation condition allows easy evaluation of the scalars: for each integer m, we have $u(m) = c(m)$ since all translates $\phi(m-n)$ other than $\phi(m-m)$ contribute nothing to the sum at m. Thus, any function in a sampling space is completely determined by its *samples*, or values at the integers. Additional hypotheses may also be imposed on the values of $\phi(x)$ at non-integer $x \in \mathbf{R}$, to obtain additional properties for the sampling space and its functions.

For a given sequence of scalars $\{c(n) : n \in \mathbf{Z}\}$, the following partial sum belongs to span $\{\phi(x-n) : n \in \mathbf{Z}\}$ for any integers $M \leq N$:

$$u_{MN}(x) \stackrel{\text{def}}{=} \sum_{n=M}^{N} c(n)\phi(x-n) \quad (4.19)$$

We need some notion of convergence to a limit in order to make sense of the infinite superpositions available in a sampling space:

Definition 3 *Let* $\mathbf{B} \stackrel{\text{def}}{=} \{\phi_n : n \in \mathbf{Z}\}$ *be a set of functions of one real variable, and suppose that* span $\mathbf{B}$ *forms a vector space with norm* $\|\cdot\|$. *The* closure of span $\mathbf{B}$ in the norm $\|\cdot\|$, *denoted* $\overline{\text{span}}\,\mathbf{B}$, *consists of all expressions of the form*

$$u(x) = \sum_{n=-\infty}^{\infty} c(n)\phi_n(x),$$

where the scalars $\{c(n) : n \in \mathbf{Z}\}$ *produce partial sums* u_{MN} *with the property that for every* $\epsilon > 0$, *there is some number* $T_\epsilon > 0$ *such that* $\|u_{MN}\| < \epsilon$ *whenever* $N \geq M \geq T_\epsilon$ *or* $M \leq N \leq -T_\epsilon$.

In other words, the tails of infinite superpositions contribute arbitrarily small errors as measured by the norm. Addition and scalar multiplication are defined componentwise for such expressions:

$$\sum_{n=-\infty}^{\infty} c(n)\phi_n(x) + a \sum_{n=-\infty}^{\infty} c'(n)\phi_n(x) \stackrel{\text{def}}{=} \sum_{n=-\infty}^{\infty} [c(n) + ac'(n)]\phi_n(x). \tag{4.20}$$

A norm may be defined on the resulting vector space by taking limits of the norms of partial sums:

$$\|u\| \stackrel{\text{def}}{=} \lim_{-M,N\to\infty} \|u_{MN}\|,$$

and we are sure that this limit exists because both $\{\|u_{0N}\| : N > 0\}$ and $\{\|u_{-M0}\| : M > 0\}$ are Cauchy sequences. Indeed, the construction of the normed vector space $\overline{\text{span}}\,\mathbf{B}$ is analogous to the construction of real numbers from Cauchy sequences of rational numbers. There is even a version of completeness similar to that of Theorem 1.9:

Theorem 4.7 *Given a set* $\mathbf{B} = \{\phi_n : n \in \mathbf{Z}\}$ *and a norm* $\|\cdot\|$, *form* $\overline{\text{span}}\,\mathbf{B}$. *Then for any* $\mathbf{V} = \{\psi_n : n \in \mathbf{Z}\} \subset \overline{\text{span}}\,\mathbf{B}$, *we have* $\overline{\text{span}}\,\mathbf{V} \subset \overline{\text{span}}\,\mathbf{B}$. □

Given an interpolating function ϕ and a norm $\|\cdot\|$, we get a sampling space $\overline{\text{span}}\,\mathbf{B}$ from the set $\mathbf{B} = \{\phi_n(x) = \phi(x - n) : n \in \mathbf{Z}\}$. The choice of ϕ and the choice of norm both influence the sampling space. For example, fix $\phi(x) = \mathbf{1}(x)$, the characteristic function of $[0, 1)$, and consider the sampling space generated by $\mathbf{B} = \{\mathbf{1}(x - n) : n \in \mathbf{Z}\}$ by the closure in three norms. $\mathbf{1}$ evidently satisfies the interpolation condition,[1] and the partial sum u_{MN} will be a piecewise constant function:

$$u_{MN}(x) = \begin{cases} c(n), & \text{if } x \in [n, n+1) \text{ for some } n \in \{M, M+1, \ldots, N-1, N\}; \\ 0, & \text{if } x \notin [M, N+1). \end{cases}$$

We compute the tail errors in three norms:

[1] To get continuity at the integer sample points, use a balanced support interval $I = [-\frac{1}{2}, \frac{1}{2}]$ and put $\phi = \mathbf{1}_I$.

1. Let $\|u\|_1 \stackrel{\text{def}}{=} \int_{\mathbf{R}} |u(x)|\,dx$. Then $\|u_{MN}\|_1 = \sum_{n=M}^{N} |c(n)|$, so the expression $u(x) = \sum_n c(n)\mathbf{1}(x-n)$ belongs to $\overline{\text{span}}\,\mathbf{B}$ if and only if $\sum_n |c(n)|$ is a convergent series.

2. Let $\|u\|_2 \stackrel{\text{def}}{=} \left(\int_{\mathbf{R}} |u(x)|^2\,dx\right)^{1/2}$. Then $\|u_{MN}\|_2 = \left(\sum_{n=M}^{N} |c(n)|^2\right)^{1/2}$ is the ℓ^2 norm of the partial sequence $\{c(n) : M \le n \le N\}$, so the expression $u(x) = \sum_n c(n)\mathbf{1}(x-n)$ belongs to $\overline{\text{span}}\,\mathbf{B}$ if and only if $c \in \ell^2(\mathbf{Z})$, if and only if $\sum_n |c(n)|^2$ is a convergent series.

∞. Let $\|u\|_\infty \stackrel{\text{def}}{=} \max\{|u(x)| : x \in \mathbf{R}\}$. Then $\|u_{MN}\|_\infty = \max\{|c(n)| : M \le n \le N\}$, so the expression $u(x) = \sum_n c(n)\mathbf{1}(x-n)$ belongs to $\overline{\text{span}}\,\mathbf{B}$ if and only if $c(n) \to 0$ as $|n| \to \infty$.

Example 2 deserves special attention since the norm $\|\cdot\|_2$ is derived from the inner product in $L^2(\mathbf{R})$, and the translates $\mathbf{B} = \{\mathbf{1}(x-n) : n \in \mathbf{Z}\}$ form an orthonormal set in that inner product space. In general, when the L^2 norm is chosen, and ϕ generates an orthonormal set $\mathbf{B}$, then by adapting the proof of Parseval's formula[2] we can show that u belongs to $\overline{\text{span}}\,\mathbf{B}$ if and only if the sequence of samples $\{u(n) : n \in \mathbf{Z}\}$ belongs to $\ell^2(\mathbf{Z})$. Furthermore, $\mathbf{B}$ will be an orthonormal Schauder basis for $\overline{\text{span}}\,\mathbf{B}$, and since $\langle \phi_n, \phi_m \rangle = \delta(n-m)$ for $n, m \in \mathbf{Z}$, we see that

$$\int_{-\infty}^{\infty} \bar{\phi}(x-n)u(x)\,dx = \langle \phi_n, u \rangle = \sum_m u(m)\,\langle \phi_n, \phi_m \rangle = u(n). \tag{4.21}$$

This shows how we can compute the expansion coefficients of u in the orthonormal basis $\mathbf{B}$ by sampling, a simpler alternative to inner product integrals. However, the inner product must be used to evaluate the initial approximation of an arbitrary function $u \in L^2(\mathbf{R})$ in $\overline{\text{span}}\,\mathbf{B}$:

$$P : L^2(\mathbf{R}) \to \overline{\text{span}}\,\mathbf{B}; \qquad Pu(x) \stackrel{\text{def}}{=} \sum_{n \in \mathbf{Z}} \langle \phi_n, u \rangle\, \phi_n(x). \tag{4.22}$$

This P is actually an orthogonal projection, generalizing the one in Exercise 13 of Chapter 2. It is not true that $\langle \phi_n, u \rangle = u(n)$ for arbitrary $u \in L^2(\mathbf{R})$, but only that $\langle \phi_n, u \rangle = \langle P\phi_n, u \rangle = \langle \phi_n, Pu \rangle = Pu(n)$ since $P\phi_n = \phi_n$ and $P = P^*$.

We now restrict our attention to the L^2 norm, $\|\cdot\|_2$, and consider the sampling spaces generated by two other interpolating functions ϕ. The first is a hat function $\phi(x) = h(x)$ like one of those defined in Equation 2.22:

$$h(x) \stackrel{\text{def}}{=} \begin{cases} 1 - |x|, & \text{if } |x| < 1; \\ 0, & \text{if } x \le -1 \text{ or } x \ge 1, \end{cases} \tag{4.23}$$

This bounded, continuous function has compact support in the interval $[-1, 1]$, so it is square-integrable. The sampling space generated by h consists of all piecewise

[2] Solve exercise 14 of Chapter 2!

linear square-integrable functions with joints at the integers. Although the translates $\mathbf{B} = \{h(x-n) : n \in \mathbf{Z}\}$ are not orthogonal, it is still true that $\sum_n c(n)h(x-n)$ belongs to the sampling space $\overline{\text{span}}\,\mathbf{B}$ if and only if the infinite series $\sum_n |c(n)|^2$ converges. This is left as an exercise.

Band-limited functions

The second example is $\phi(x) = \text{sinc}\,(x) = \frac{\sin(\pi x)}{\pi x}$, as introduced in Equation 3.47 and graphed in Figure 3.7. Recall that the Fourier integral transform of this ϕ is the indicator function of the balanced interval $I = [-\frac{1}{2}, \frac{1}{2}]$, so $\mathcal{F}\phi = \mathbf{1}_I$ is piecewise constant and compactly supported on I. It satisfies the interpolation condition because the numerator of $\text{sinc}\,(x)$ is zero at every nonzero $x \in \mathbf{Z}$.

Plancherel's theorem may be used to show that the set of functions $\mathbf{B} = \{\phi_n(x) \stackrel{\text{def}}{=} \text{sinc}\,(x-n) : n \in \mathbf{Z}\}$ is an orthonormal basis for $\overline{\text{span}}\,\mathbf{B}$:

$$\langle \phi_n, \phi_m \rangle = \langle \mathcal{F}\phi_n, \mathcal{F}\phi_m \rangle = \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{-2\pi i(n-m)\xi}\, d\xi = \delta(n-m),$$

using the identity $\mathcal{F}\phi_k(\xi) = e^{-2\pi ik\xi}\mathcal{F}\phi(\xi)$, whose proof was left as an exercise. Thus the expansion coefficients of $u \in V$ are just the samples $\{u(n) : n \in \mathbf{Z}\}$. But the same identity shows that every $u \in V$ has a Fourier integral transform supported in I as well:

$$\mathcal{F}u(\xi) = \sum_n u(n)\mathcal{F}\phi_n(\xi) = \mathbf{1}_I(\xi)\left(\sum_n u(n)e^{-2\pi in\xi}\right), \tag{4.24}$$

which is evidently the restriction to I of the 1-periodic function given by the exponential Fourier series in parentheses.

A function $u = u(x)$ will be called *band limited* with *bandwidth* T if its Fourier integral transform is a square-integrable function $s = s(\xi)$ satisfying the compact support condition $s(\xi) = 0$ if $\xi < -\frac{T}{2}$ or $\xi > \frac{T}{2}$. This s is called the *spectral density* of u. The sampling space for sinc is thus a subset of the band-limited functions u with bandwidth 1. In fact, it is all of them, and we will show more generally that the sampling space generated by $\phi(x) = T\text{sinc}\,(Tx)$ consists of all band-limited functions of bandwidth T.

We begin by showing that band-limited functions must be very smooth:

Theorem 4.8 *If $u = u(x)$ is band-limited, then*

1. *For every $n \geq 0$, $\frac{d^n}{dx^n}u(x)$ exists at each $x \in \mathbf{R}$. In particular, $u^{(n)}(0)$ exists.*
2. *The Taylor series for u about $x = 0$ converges to u everywhere:*

$$u(x) = \sum_{n=0}^{\infty} \frac{u^{(n)}(0)}{n!}x^n.$$

Proof: Let T be the bandwidth of u, and let $s = \mathcal{F}u$ be its spectral density. Fix $x_0 \in \mathbf{R}$ and write

$$u(x_0) = \mathcal{F}^{-1}s(x_0) = \int_{-\infty}^{\infty} s(\xi)e^{2\pi i x_0 \xi}\, d\xi = \int_{-\frac{T}{2}}^{\frac{T}{2}} s(\xi)e^{2\pi i x_0 \xi}\, d\xi.$$

The domain of integration is $I = [-\frac{T}{2}, \frac{T}{2}]$ because $s = 0$ outside that interval. The first result is obtained by differentiating under the integral sign:

$$\frac{d^n}{dx^n}u(x_0) = \int_{-\frac{T}{2}}^{\frac{T}{2}} (2\pi i\xi)^n\, s(\xi)e^{2\pi i x_0 \xi}\, d\xi,$$

We may interpret this as an inner product in $L^2(I)$, and estimate it with the Cauchy–Schwarz inequality for the derived norm:

$$\left|\frac{d^n}{dx^n}u(x_0)\right| \leq \|(2\pi\xi)^n\|\,\|s\| \leq \frac{|\pi T|^{n+\frac{1}{2}}}{\sqrt{n+\frac{1}{2}}}\|u\|.$$

Here we used the identities $|i| = 1$ and $|e^{2\pi i x_0 \xi}| = 1$ for all x_0 and ξ. Also, $\|s\| = \|\mathcal{F}u\| = \|u\|$ by Plancherel's theorem. For any $n \geq 0$, this gives $|u^{(n)}(x_0)| \leq a|\pi T|^n$, where $a = \sqrt{2\pi T}\,\|u\| \geq 0$ is a constant independent of n or x_0. In particular, the inequality holds for $x_0 = 0$. Thus for all $x \in \mathbf{R}$ and all $n \geq 0$,

$$\left|\frac{u^{(n)}(0)}{n!}x^n\right| \leq a\frac{|\pi T x|^n}{n!},$$

where a is independent of $n \geq 0$ and $x \in \mathbf{R}$. Using this in the comparison test with the absolutely convergent Taylor series for $ae^{|\pi T x|}$ gives the second result. □

Since the sequence of numbers $\{u^{(n)}(0) : n = 0, 1, 2, \ldots\}$ determines all the values $\{u(x) : x \in \mathbf{R}\}$, it seems less miraculous that samples of a band-limited u, at regularly-spaced points sufficiently close together, also determine $u(x)$ everywhere:

Theorem 4.9 (Shannon) *If $u = u(x)$ is band-limited with bandwidth T, then*

$$u(x) = \sum_{n=-\infty}^{\infty} u\left(\frac{n}{T}\right)\operatorname{sinc}(Tx - n).$$

Proof: Let s be the spectral density of u that is supported on the interval $J = [-\frac{T}{2}, \frac{T}{2}]$. Then $s(\xi) = \mathbf{1}_J(\xi)s(\xi)$ for all $\xi \in \mathbf{R}$, since s is already localized to J. We expand s in its T-periodic Fourier series:

$$s(\xi) = \mathbf{1}_J(\xi)s(\xi) = \mathbf{1}_J(\xi)\sum_{n=-\infty}^{\infty} c(n)e^{2\pi i n\xi/T} = \sum_{n=-\infty}^{\infty} c(n)b_n(\xi). \tag{4.25}$$

Figure 4.1: Left: Part of the graph of $e^{-x^2}\mathrm{sinc}\,(x)$. Right: Part of the graph of the "hat" function. Both curves have maximum value 1, and two roots nearest 0 at $x = -1$ and $x = +1$.

The factor $\mathbf{1}_J(\xi)$ is needed for equality on all of $\mathbf{R}$ since the series defines a periodic function supported outside the interval J. We combined it with the exponential to get

$$b_n(\xi) \stackrel{\text{def}}{=} \mathbf{1}_J(\xi)e^{2\pi in\xi/T} = \mathbf{1}_I(\xi/T)e^{2\pi in\xi/T},$$

where $I = [-\frac{1}{2}, \frac{1}{2}]$ and $\mathcal{F}^{-1}\mathbf{1}_I(x) = \mathrm{sinc}\,(x)$. By Exercises 11 and 12 of Chapter 3, we may evaluate $\mathcal{F}^{-1}b_n(x) = T\mathrm{sinc}\,(Tx + n)$. We now reverse course and write u as the inverse Fourier integral transform of the series for s:

$$u(x) = \mathcal{F}^{-1}s(x) = \sum_{n=-\infty}^{\infty} c(n)\mathcal{F}^{-1}b_n(x) = \sum_{n=-\infty}^{\infty} c(n)T\mathrm{sinc}\,(Tx + n). \tag{4.26}$$

On the other hand, the Fourier coefficients $\{c(n)\}$ for Equation 4.25 can also be written as inverse Fourier integral transforms of s:

$$c(n) = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} s(\xi)e^{-2\pi in\xi/T}\,d\xi = \frac{1}{T}\int_{-\infty}^{\infty} s(\xi)e^{2\pi i(-\frac{n}{T})\xi}\,d\xi = \frac{1}{T}u\left(-\frac{n}{T}\right). \tag{4.27}$$

Substituting Equation 4.27 into Equation 4.26 and replacing $n \leftarrow -n$ yields the result. □

There are benefits to choosing an interpolating function ϕ with fast decay away from its peak at 0. We may wish to ignore entirely the values of ϕ outside $I = [-1, 1]$, and then faster decay implies a smaller error $\|\phi - \mathbf{1}_I\phi\|^2$. This quantity is relatively large for sinc, about 10%. It can be reduced to about 0.1% by using the *Gaussian sinc function* $\phi(x) = e^{-x^2}\mathrm{sinc}\,(x)$, which is depicted in Figure 4.1 together with the hat function of Equation 4.23, to emphasize their similarity. The Gaussian sinc function does not generate an orthonormal basis for its sampling space, however.

4.2 Measurement and Estimation

Consider what it means to make a measurement. For example, what is the length of a pencil? We must suppose that the pencil actually has a length that various observers can agree on. Upon close examination, even this simple problem is surprisingly complex. It is known that a pencil expands and contracts with temperature and humidity changes. Also, the end of a pencil looks rough under a microscope, so the measurement points must be specified among the peaks and valleys at the ends. Indeed, any mechanical device that touches the rough end of a pencil will press down some of these peaks, influencing the measurement and causing different observations to disagree.

In general, it is common to suppose that an observable quantity has an *ideal value*, which is a constant real number. In practice, there will be various errors and uncertainties in any physical measurement. Taking a measurement at best gives some information about the likelihood of different ideal values. One way to model this is to introduce a random variable whose probability density function is determined by a measurement, and then estimate the ideal value by some expected value[3] or other functional of the density. This estimated ideal value should depend on the measurement, of course, or else the instrument is useless. We make this precise, and derive formulas, using the notion of joint and conditional probabilities.

4.2.1 Quantization, precision, and accuracy

In practice, measurements are always *quantized*, taking one of finitely many discrete *representable values* that the instrument can produce. For example, a twelve-inch ruler marked in sixteenths of an inch can yield one of only $16 \times 12 = 192$ different nonzero measurements $x \in \{\frac{n}{16} : n = 1, \ldots, 192\}$. Some measured lengths would fall between marks, of course, and these would be read to the nearest sixteenth of an inch. The *quantization error* of any quantized measurement is half the distance between adjacent values. For this ruler, that is $\frac{1}{32}$ inch. In some cases, however, it is safe to assume an *ideal instrument*, with zero quantization error, because the quantization error is negligible compared to other errors.

A length measurement x with the example ruler implies only that the ideal value y lies in the interval $I_x \stackrel{\text{def}}{=} [x - \frac{1}{32}, x + \frac{1}{32}]$. Told that the ruler read x, we may only conclude that $y \in I_x$ and that every value in that interval is equally likely. Hence, we may introduce a random variable $Y = Y_x$ to represent our knowledge, and give it a uniform probability density function supported on that interval.

In general, a measurement x yields only partial information about the ideal value, which may therefore be represented by the probability density function of a random variable $Y = Y_x$. The variance of Y indicates the uncertainty of this information, so we define the *imprecision* of the measurement to be the square root of this variance. In terms of expectations E, this is the root-mean-square error $\sqrt{E(|Y - E(Y)|^2)}$. Note that even an ideal instrument can have nonzero

[3]Some random variables have densities with no expected value!

imprecision. Conversely, a non-ideal instrument can have zero imprecision. An instrument that counts pennies can make only nonnegative integer measurements, and has quantization error $\frac{1}{2}$, but can have zero imprecision if it never miscounts since the ideal value is also a nonnegative integer.

Given a measurement x of an ideal value y, we may hope that x is close to y. Since our knowledge of y is limited, we may at best label, or *calibrate*, the measuring instrument so that $x = E(Y_x)$, the mean value of the random variable Y_x determined by the measurement x.

But even if the instrument is not calibrated, we may define the *inaccuracy* of the measurement as the root-mean-square error $\sqrt{E(|Y - y|^2)}$. It can be shown, using the Cauchy–Schwarz inequality of Lemma 2.4, that this is minimized when $y = E(Y)$, and thus inaccuracy is never smaller than imprecision.

Measurement density functions

Inaccuracy and imprecision may be computed from the *measurement density function*, which is a nonnegative function $f = f(x, y)$ giving the likelihood of any combinations of ideal value y and measured value x:

$$\Pr(X \in [a_X, b_X], Y \in [a_Y, b_Y]) \stackrel{\text{def}}{=} \int_{a_X}^{b_X} \int_{a_Y}^{b_Y} f(x, y)\, dx dy. \tag{4.28}$$

Since $\Pr(X \in \mathbf{R}, Y \in \mathbf{R}) = 1$, we must have that $\iint_{R^2} f(x, y)\, dx dy = 1$. Such an f is called a *joint probability density function* for the random variables X and Y, giving likelihoods that they fall within particular ranges. With another normalization, it can be used to compute the likelihood that, given a measurement x, the random variable $Y = Y_x$ representing the ideal value falls in a particular range $[a_Y, b_Y]$:

$$\Pr(Y \in [a_Y, b_Y] \,|\, x) \stackrel{\text{def}}{=} \int_{a_Y}^{b_Y} f(y \,|\, x)\, dy, \tag{4.29}$$

where

$$f(y \,|\, x) \stackrel{\text{def}}{=} \frac{1}{c_x} f(x, y); \qquad c_x \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(x, y)\, dy. \tag{4.30}$$

The normalizing constant c_x must be finite and positive at each representable x; it guarantees that $\Pr(Y \in \mathbf{R} \,|\, x) = 1$. Such a normalized $f(y \,|\, x)$ is called a *conditional probability density function.* If in addition the instrument has finite imprecision, then the variance of Y given x will be finite for each representable x. This variance is comparable to $\int_{-\infty}^{\infty} y^2 f(y \,|\, x)\, dy$, which in turn is just a multiple of $\int_{-\infty}^{\infty} y^2 f(x, y)\, dy$. If this last integral is finite, we will say that the instrument is *focused.*

An instrument with measurement density f is *calibrated* if, for each measurement x,

$$E(Y_x) = E(Y \,|\, x) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} y f(y \,|\, x)\, dy = x.$$

The measurement density may be uncalibrated by an amount that depends on x. For the example ruler, the domain of the measurement is $\{\frac{0}{16}; \frac{1}{16}, \ldots, \frac{192}{16}\}$, and the measurement density function is:

$$f(x,y) = \begin{cases} \frac{1}{12}, & \text{if } x = 0 \text{ and } 0 \le y < \frac{1}{32}; \\ \frac{1}{12}, & \text{if } x \in \{\frac{1}{16}, \ldots, \frac{191}{16}\} \text{ and } x - \frac{1}{32} \le y < x + \frac{1}{32}; \\ \frac{1}{12}, & \text{if } x = 12 \text{ and } 12 - \frac{1}{32} \le y \le 12; \\ 0, & \text{otherwise.} \end{cases}$$

Then nonzero length measurements are calibrated, but for $x = 0$ we have $E(Y_0) = \frac{1}{64} \neq 0$, and for $x = 12$ we have $E(Y_{12}) = 12 - \frac{1}{64} \neq 12$.

The measurement density may also be uncalibrated by an unknown amount that is not computable from x. For example, let Y be the number of photons of all colors impinging on a detector in one second, and x the number that the detector counts. If 90% of red photons but only 50% of blue photons are detected, then the measurement density will be uncalibrated by an amount that depends on the color spectrum. For blue light, we will have $E(Y_x) = x/0.50$, while for red light it will be $E(Y_x) = x/0.90$.

The imprecision of an instrument at a measurement x is the square root of the variance of Y_x. This is finite for focused densities. For a calibrated measurement density f, variance is given by the following formula:

$$\mathrm{Var}(Y_x) = \mathrm{Var}(Y \mid x) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} |y - x|^2 f(y \mid x)\, dy. \tag{4.31}$$

Then the imprecision is $\Delta Y_x = \sqrt{\mathrm{Var}(Y \mid x)}$.

An alternative measure of imprecision is the width of the smallest interval centered at $E(Y)$ that has at least 50% probability of holding the value of Y. For a calibrated measurement, this is given by the following formula:

$$W(Y \mid x) \stackrel{\text{def}}{=} \min \left\{ \delta > 0 : \int_{x-\delta/2}^{x+\delta/2} f(y \mid x)\, dy \ge 1/2 \right\}. \tag{4.32}$$

The imprecision by this definition is $\Delta Y_x = W(Y \mid x)$.

The measurement density function is usually impossible to determine. However, if the measurement error is a sum of many small independent errors, the Central Limit Theorem[4] justifies approximating it by a *normal density*:

$$f(x,y) = \frac{1}{\sqrt{2\pi}\,\sigma|b-a|} \exp\left(-\frac{(x-y)^2}{2\sigma^2}\right), \qquad x \in [a,b],\ y \in \mathbf{R} \tag{4.33}$$

where the parameter $\sigma > 0$ is chosen to fit the imprecision of the instrument, and the range of possible values x is limited to allow normalization. The normalizing constants will be $c_x = 1$ for all x. This model implies that each measurement is calibrated and has the same imprecision: it is exactly σ by the variance criterion, and approximately 0.954σ by the 50%-interval criterion.

[4]See Ross, *A First Course in Probability*, third edition, page 340.

Signal to noise ratio

Inaccuracy in measurement is often called *noise*, especially in sequences of measurements such as digitally recorded sounds or images. The ideal values in this case are called the *signal*.

Let $u = \{u(n) : n \in \mathbf{N}\}$ be a sequence of measurements, and suppose that each value $u(n)$ can be written as $v(n) + q(n)$, where $v(n)$ is the ideal value and $q(n)$ is a measurement error. The quality at sample n is just the squared ratio $|v(n)|^2/|q(n)|^2$, which is called the *signal to noise ratio*. But it can be misleading to focus on a single sample, and the measurements actually produce two sequences v and q. A useful concept is *local power*, or $\|v\|^2_{[a,b]}$, defined by

$$\|v\|^2_{[a,b]} \stackrel{\text{def}}{=} \frac{1}{1+b-a} \sum_{n=a}^{b} |v(n)|^2, \tag{4.34}$$

which is the average energy, or square norm, per sample in the sampling interval $[a, b]$. The signal to noise ratio of a sequence u defined on $[a, b]$ expressed using local power is therefore

$$SNR(u) \stackrel{\text{def}}{=} 10 \log_{10} \frac{\|v\|^2_{[a,b]}}{\|q\|^2_{[a,b]}}. \tag{4.35}$$

Local power can be used to compute the signal to noise ratio of functions $f = f(t)$ of a real variable in the interval $I = [0, 1]$. For each $N \in \mathbf{Z}^+$, the samples $v_N(n) = f(n/N)$, $1 \leq\leq n \leq N$, when used in Equation 4.34, give a Riemann sum approximation to $\int_0^1 |f(t)|^2\, dt = \|f\|^2$, the square norm of f in $L^2(I)$. If f is continuous on I, then $\lim_{N\to\infty} \|v_N\|^2_{[1,N]} = \|f\|^2$ exists. We may therefore define the signal to noise ratio for such functions as

$$SNR(f) \stackrel{\text{def}}{=} 10 \log_{10} \frac{\|f_s\|^2}{\|f_q\|^2}, \tag{4.36}$$

where $f = f_s + f_q$ is written as a sum of ideal signal f_s and noise f_q, with both parts required to be in $L^2(I)$.

Of course, these notions are useful only when the signal and noise sequences are known. That is the case when the noise is caused by some transformation of an exactly-known initial sequence; then q is the difference between the initial and final sequences. When q is unknown, is can sometimes be estimated or modeled as a sequence of random variables with finite means and variances. So-called *white Gaussian noise* is a sequence of independent random variables[5] which each have the Gaussian, or normal, probability density function $f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)$ with mean 0 and variance σ^2.

[5]When the random variables take just the two values T and F, with $\Pr(T) = p$ and $\Pr(F) = 1 - p$ for all of them given by some fixed $0 < p < 1$, the result is called a sequence of Bernoulli trials.

4.2.2 Estimation

In our model, it is not possible to determine the ideal value y from a single measurement x since all we get is a random variable Y_x. We imagine that many ideal values might yield the same measurement, each with some likelihood, and this many-to-one relationship is not invertible. But x and y play similar roles in the measurement density function $f = f(x, y)$, and we may also imagine that an ideal value y might result in various measurements X_y, each with its own likelihood. For an ideal instrument, the likelihoods are

$$\Pr(X \in [a, b] \,|\, y) \stackrel{\text{def}}{=} \int_a^b f(x \,|\, y)\, dx, \tag{4.37}$$

where the integrand is a normalized version of the measurement density function called the *conditional measurement density*:

$$f(x \,|\, y) \stackrel{\text{def}}{=} \frac{1}{c_y} f(x, y); \qquad c_y \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(x, y)\, dx. \tag{4.38}$$

For each y, the normalizing constant c_y guarantees $\Pr(X \in \mathbf{R} \,|\, y) = 1$.

The normal density of Equation 4.33 is a common model for estimating unknown measurement density functions, though its definition must be slightly modified:

$$f(x, y) = \frac{1}{\sqrt{2\pi}\,\sigma |b - a|} \exp\left(-\frac{(x - y)^2}{2\sigma^2}\right), \qquad x \in \mathbf{R},\ y \in [a, b]. \tag{4.39}$$

From the estimation perspective, the range of ideal values y is constrained to a bounded interval, but we imagine that a measurement X_y can produce any real number.

By repeated measurement, the likelihoods of various measurements X can be approximated, and then the one-to-many relationship $y \mapsto X_y$ can be inverted to get a good approximation for y. Such an approximate inverse is called an *estimator*, and it is implemented as a *decision function*, $y \approx d(x_1, \ldots, x_N)$, giving an approximation for the ideal value y from a sequence of N independent measurements $x_1, \ldots, x_N$. For example, $d_2^a(x_1, x_2) \stackrel{\text{def}}{=} \frac{1}{2}(x_1 + x_2)$ is one possible estimator: it uses the average of two independent measurements as an estimate for the ideal value. More generally, we may take the average of N measurements,

$$d_N^a(x_1, \ldots, x_N) \stackrel{\text{def}}{=} \frac{x_1 + \cdots + x_N}{N}, \tag{4.40}$$

which under certain circumstances converges to the ideal value as $N \to \infty$.

We will always assume that the N measurements $x_1, \ldots, x_N$ are independent, so that likelihood calculations factor into N pieces:

$$\begin{aligned} &\Pr(X_1 \in [a_1, b_1], \ldots, X_N \in [a_N, b_N] \,|\, y) \;= \\ &\quad = \int_{a_1}^{b_1} \cdots \int_{a_N}^{b_N} f(x_1, \ldots, x_N \,|\, y)\, dx_1 \cdots dx_N \end{aligned}$$

$$= \left(\int_{a_1}^{b_1} f(x_1 \,|\, y)\, dx_1\right) \cdots \left(\int_{a_N}^{b_N} f(x_N \,|\, y)\, dx_N\right),$$

since $f(x_1, \ldots, x_N \,|\, y) \stackrel{\text{def}}{=} f(x_1 \,|\, y) \cdots f(x_N \,|\, y) = c_y^{-N} f(x_1, y) \cdots f(x_N, y)$. The expected value of any decision function $d = d(x_1, \ldots, x_N)$ for y is then a multiple integral involving N copies of the normalized measurement density function:

$$E(d \,|\, y) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} d(x_1, \ldots, x_N) f(x_1 \,|\, y) \cdots f(x_N \,|\, y)\, dx_1 \cdots dx_N.$$

Here we consider all possible sequences of measurements $x_1, \ldots, x_N$, weighted by their likelihoods. We can sometimes design a decision function d so that its expected value is the ideal value: $E(d \,|\, y) = y$. Such a d is called an *unbiased estimator*. The averaging estimator d_N^a is one example. A biased estimator may also be useful, especially if we know its bias. For example, consider measuring the ideal breaking strength y of a rope by hanging various weights $x_1, \ldots, x_N$ on it. Our instrument will record the strength measurement x_k if and only if $x_k < y$. Choosing units so that all values lie in the interval $I = [0, 1]$, we get a measurement density function on the unit square $0 \le x, y \le 1$ defined by the formula

$$f(x, y) = \begin{cases} 2, & \text{if } 0 \le x < y \le 1; \\ 0, & \text{otherwise.} \end{cases} \tag{4.41}$$

The factor 2 insures that $\iint_{I \times I} f(x, y)\, dxdy = 1$. Given N measurements $x_1, \ldots, x_N$ with this instrument, we let $d(x_1, \ldots, x_N) \stackrel{\text{def}}{=} \max\{x_1, \ldots, x_N\}$ be the estimator for the ideal value y. We are sure that $d(x_1, \ldots, x_N) < y$ since $x_k < y$ for all k, so the estimator is biased on the low side of y. But this bias can be removed by using $d = \frac{N+1}{N} \max\{x_1, \ldots, x_N\}$ since for any $y > 0$,

$$\begin{aligned} E(d(X_1, \ldots, X_N) \,|\, y) \;&= \\ &= \int_I \cdots \int_I d(x_1, \ldots, x_N) f(x_1 \,|\, y) \cdots f(x_N \,|\, y)\, dx_1 \cdots dx_N \\ &= (2y)^{-N} \int_I \cdots \int_I \max\{x_1, \ldots, x_N\} f(x_1, y) \cdots f(x_N, y)\, dx_1 \cdots dx_N \\ &= 2^N (2y)^{-N} N! \int_0^y dx_1 \int_0^{x_1} dx_2 \cdots \int_0^{x_{N-1}} x_1\, dx_N \\ &= y^{-N} N! \int_0^y x_1 \frac{x_1^{N-1}}{(N-1)!}\, dx_1 \;=\; \frac{Ny}{N+1} \;<\; y. \end{aligned}$$

The factor $N!$ appears on the last line since there are $N!$ orderings of $(x_1, \ldots, x_N)$, and each one contributes the same amount to the total as the single one $0 \le x_N < \cdots < x_1 < y$ that we actually integrate. Iterating the $N - 1$ inner integrals successively produces:

$$x_{N-1}, \frac{x_{N-2}^2}{2!}, \ldots, \frac{x_1^{N-1}}{(N-1)!}, \tag{4.42}$$

and the remaining steps are straightforward. It is left as an exercise to show that the normalizing constant that determines $f(x \mid y)$ from $f(x, y)$ is $c_y = 2y$.

The inaccuracy $R(d, y)$ of an estimate d for an ideal value y, which is called *risk*, is similar to the inaccuracy of a measurement: it is given by the formula

$$\begin{aligned} R^2(d, y) &= E\left(|d - y|^2\right) \\ &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} |d(x_1, \ldots, x_N) - y|^2 f(x_1 \mid y) \cdots f(x_N \mid y)\, dx_1 \cdots dx_N. \end{aligned} \tag{4.43}$$

For an unbiased estimator, the risk is the square root of the variance $\mathrm{Var}(d \mid y)$, which is like the imprecision of an instrument rather than the possibly larger inaccuracy. For example, if $d(x) = x$ and f is the calibrated normal density of Equation 4.39 with imprecision σ, then $\mathrm{Var}(d(X)) = \sigma^2$ and the risk $R(d, y) = \sigma$ of the estimate is exactly the same as the imprecision of the measurement. It is also independent of y, just like the imprecision. For the same measurement density, averaging N independent measurements with the averaging estimator d_N^a of Equation 4.40 decreases the risk:

$$\mathrm{Var}(d_N^a) = \frac{\sigma^2}{N} \quad \Rightarrow R(d_N^a, y) = \frac{1}{\sqrt{N}} R(d, y). \tag{4.44}$$

This is another consequence of the Central Limit Theorem.

The Cramér-Rao lower bound

Can the risk be reduced faster by a more clever choice of decision function? Any estimate $d(x_1, \ldots, x_N)$ gets some information about the density f from the measurements $x_1, \ldots, x_N$, but there are many ways to combine these measurements to compute y. Also, the risk decreases with more samples N, if the approximate density determined from the histogram of $x_1, \ldots, x_N$ converges to $f(x \mid y)$ as $N \to \infty$. Still, the computation of y from this density is possible if and only if $f(x \mid y)$ changes with y. For instruments whose measurements depend smoothly on the ideal value y, that holds if and only if $\frac{\partial}{\partial y} f(x \mid y)$ is nonzero. Thus N and $\frac{\partial}{\partial y} f(x \mid y)$ may be expected to take part in any risk estimate.

But in fact, these two are the only relevant factors. Regardless of the choice of decision function d_N for N independent measurements, there is a lower bound on $\mathrm{Var}(d_N \mid y))$ that decreases with N and depends only on how the measurement density $f(x \mid y)$ changes with y:

$$\mathrm{Var}(d_N \mid y) \geq \frac{1}{N\, E\left(\left[\frac{\partial}{\partial y} \log f(x \mid y)\right]^2\right)}, \tag{4.45}$$

where we take $\frac{\partial}{\partial y} \log f(x \mid y) = 0$ if $f(x \mid y) = 0$, by convention. Inequality 4.45, which is proved as Theorem 1 on page 18 of Hoel, Port and Stone's *Introduction to Statistical Theory*, is known as the *Cramér-Rao lower bound*. It implies that the

risk of an unbiased estimate can be reduced by at most a factor $1/\sqrt{N}$ by taking N independent measurements.

This lower bound is attained in many usual cases. Using the averaging estimator of Equation 4.40 with the normal measurement density of Equation 4.39 gives $\mathrm{Var}(d_N^a, y) = \sigma^2/N$ by the Central Limit Theorem. The Cramér-Rao lower bound is the same:

$$\frac{1}{N\,E\left(\left[\frac{\partial}{\partial y}\log f(x\,|\,y)\right]^2\right)} = \frac{\sigma^4}{N\,E\left(\left[\frac{\partial}{\partial y}(\frac{1}{2}(x-y)^2)\right]^2\right)} = \frac{\sigma^4}{N\,E\left((x-y)^2\right)} = \frac{\sigma^2}{N}.$$

However, if the estimator is biased or if the measurement density function is discontinuous or nondifferentiable in y, the Cramér-Rao inequality does not apply. It is left as an exercise to show that $R(d_N, y) = O(1/N)$ for the breaking-strength estimation of Equation 4.41. Thus, a biased estimate from a nonsmooth instrument may gain more accuracy from repeated measurements than any unbiased average using any instrument with a smooth dependence on the ideal values.

4.3 Exercises

1. Find the Lagrange polynomial through the points $(0,0)$, $(1,2)$, $(2,4)$.

2. Fix x and find a formula for the value $y = f(x)$ of the Lagrange polynomial f through the points $(-1,p)$, $(0,q)$, and $(1,r)$, in terms of x, p, q, and r. Then find dy/dq.

3. a. Find the Lagrange polynomial $\Lambda_k = \Lambda_k(x)$ of degree k that interpolates $\{(j, 2^j) : j = 0, 1, \ldots, k\}$. (Hint: use Newton's form.)

   b. Using Λ_k from part a, prove that for every $n \geq k$,

   $$\Lambda_k(n) = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{k} = \sum_{j=0}^{k} \binom{n}{j}.$$

   c. Using part b, prove that $\sum_{j=0}^{k} \binom{n}{j} = O(n^k)$, as $n \to \infty$.

4. Find the expansion in Chebyshev polynomials $T_0(x), T_1(x), T_2(x)$ of the function $f(x) = 1 - x^2$ defined for $x \in [-1, 1]$.

5. Suppose that $f(x) = mx + b$ for some constants m, b. Show that for any sampling of f, the piecewise linear approximation exactly equals f.

6. Implement `epwlinear(x,y[],a,b,N)`, equispaced piecewise linear interpolation through the set $\{(x_k, y_k) : k = 0, 1, \ldots, N\} \subset \mathbf{R}^2$, where $x_k = a + k(b - a)/N$.

7. For $x_0 = \frac{x_2 y_1 - x_1 y_2}{y_1 - y_2}$ as in Equation 4.14, prove that if $x_1 < x_2$ and $y_1 y_2 < 0$, then $x_1 < x_0 < x_2$.

8. For $x_* = \frac{1}{2}\frac{x_1^2(y_3-y_2)+x_2^2(y_1-y_3)+x_3^2(y_2-y_1)}{x_1(y_3-y_2)+x_2(y_1-y_3)+x_3(y_2-y_1)}$ as in Equation 4.16, prove that if $x_1 < x_2 < x_3$ and y_2 is not between y_1 and y_3, then $x_1 < x_* < x_3$.

9. Let $\mathbf{B} \stackrel{\text{def}}{=} \{\phi_n : n \in \mathbf{Z}\} \subset L^2(\mathbf{R})$ be an orthonormal set of functions of one real variable, and write $\|\cdot\|$ for the L^2 norm. For each $u \in L^2(\mathbf{R})$, define

$$Pu(x) \stackrel{\text{def}}{=} \sum_{n\in\mathbf{Z}} \langle \phi_n, u\rangle \, \phi_n(x).$$

Prove that $P : L^2(\mathbf{R}) \to \overline{\text{span}}\,\mathbf{B}$.

10. Suppose that $\phi(x)$ is the hat function:

$$\phi(x) \stackrel{\text{def}}{=} \begin{cases} 1-|x|, & \text{if } |x|<1; \\ 0, & \text{if } x \le -1 \text{ or } x \ge 1, \end{cases}$$

Let $c = \{c(n) : n \in \mathbf{Z}\}$ be a sequence of complex numbers, and put $f = f(t) = \sum_{n\in\mathbf{Z}} c(n)\phi(t-n)$. Show that $f \in L^2(\mathbf{R})$ if and only if $c \in \ell^2$. (Hint: compute $\|f\|^2$ in terms of $\{|c(n)|^2\}$.)

11. a. Find an example of measurement and instrument that has zero imprecision but nonzero inaccuracy.

b. Find an example of measurement and instrument that has zero imprecision and zero inaccuracy, but nonzero quantization.

12. Suppose that $f(t) = t$ for $0 \le t \le 1$. Fix $N \in \mathbf{Z}^+$ and let f_N be the piecewise constant approximation to f given by

$$f_N(t) \stackrel{\text{def}}{=} \sum_{k=0}^{N-1} f(\frac{k}{N})\mathbf{1}(Nt-k),$$

where $\mathbf{1}$ is the indicator function of the interval $[0,1)$.

a. Prove that $\|f - f_N\| = 1/3N^2$.

b. Treating f as the signal and $f - f_N$ as the noise, compute the signal to noise ratio of this approximation.

13. Let $f = f(x,y)$ be the joint probability density defined on the unit square $Q = \{(x,y) : 0 \le x, y \le 1\}$ by the formula

$$f(x,y) = \begin{cases} 2, & \text{if } 0 \le x < y \le 1; \\ 0, & \text{otherwise.} \end{cases}$$

Suppose an instrument has this f as its measurement density function. Given N measurements $x_1, \ldots, x_N$ with this instrument, let

$$d(x_1, \ldots, x_N) \stackrel{\text{def}}{=} \max\{x_1, \ldots, x_N\}$$

be the estimator for the ideal value y.

a. Prove that $\iint_Q f(x,y)\,dxdy = 1$.

b. Compute the normalizing constant c_y and determine $f(x \mid y)$.

c. Find the formula for $\Pr(X \in [a,b] \mid y)$ for $0 \leq a < b \leq 1$ d. Compute the risk $R(d,y)$ for $y > 0$.

4.4 Further Reading

- Paul G. Hoel, Sidney C. Port and Charles J. Stone. *Introduction to Statistical Theory.* Houghton Mifflin Company, Boston, 1971. ISBN 0-395-04637-8.

- Robert J. Marks, II. *Introduction to Shannon Sampling and Interpolation Theory.* Springer-Verlag, New York, 1991. ISBN 0-387-97391-5.

- John H. Mathews and Kurtis D. Fink. *Numerical Methods Using MATLAB.* Prentice-Hall, Upper Saddle River, New Jersey, third edition, 1999. ISBN 0-13-270042-5.

- Alan V. Oppenheim and Ronald W. Shafer. *Digital Signal Processing.* Prentice Hall, Englewood Cliffs, New Jersey, 1974. ISBN 0-13-214635-5.

- Sheldon Ross. *A First Course in Probability.* Macmillan Publishing Company, New York, third edition, 1988. ISBN 0-02-403850-4.

Chapter 5

Scale and Resolution

5.1 Wavelet Analysis

Suppose that $w = w(t)$ is a function of one real variable. For $a > 0$ and any real b, the new function w_{ab} defined by

$$w_{ab}(t) \stackrel{\text{def}}{=} \frac{1}{\sqrt{a}} w(\frac{t-b}{a}), \tag{5.1}$$

is a shifted and stretched copy of w. For example, if $w = \mathbf{1}$ is the indicator function of the interval $[0, 1]$, then w_{ab} is the indicator function of the interval $[b, b + a]$, divided by $\sqrt{a}$.

Now imagine that some fixed w is a waveform that is present in a signal, centered at an unknown location $t = b$ and scaled to an unknown width a. The collection $\{w_{ab} : a > 0, b \in \mathbf{R}\}$ consists of all shifted and stretched versions of w, and can be matched with the signal to determine the best values for a and b. In this context, w is called a *mother function* for the collection.

A *wavelet* $w = w(t)$ is a special kind of mother function, with three properties that make it a building block for arbitrary signals:

Duration: $|w(t)|$ is either zero or negligibly small at any t outside some bounded interval;

Cancellation: $w(t)$ oscillates so that its integral $\int_{\mathbf{R}} w(t)\, dt$ is zero;

Completeness: Any signal can be approximated by a linear combination from $\{w_{ab} : a > 0, b \in \mathbf{R}\}$.

If w belongs to the inner product space $L^2(\mathbf{R})$, where $\langle g, f \rangle = \int \bar{g}(t) f(t)\, dt$, then the first two conditions can be combined into one:

Localization: $\left| \sqrt{a} \int \bar{w}(at) w(t - b)\, dt \right| \to 0$ as $a \to +\infty$, or as $a \to 0+$, or as $b \to \pm\infty$.

We therefore say that a wavelet is *nearly orthogonal* to sufficiently distant versions of itself, since the integral that must decrease is just the inner product $\langle w_{ab}, w\rangle$, as can be seen by changing variables $t \leftarrow t + b$ and inverting the parameters $a \leftarrow 1/a$ and $b \leftarrow -b$. Also, $\|w_{ab}\| = \|w\|$ for all a, b because of the normalization by $\sqrt{a}$.

5.1.1 Haar functions

The simplest example is the *Haar wavelet*:

$$w(t) = \begin{cases} 1, & \text{if } 0 \le t < 1/2; \\ -1, & \text{if } 1/2 \le t < 1; \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Its duration is the interval $[0, 1]$ of its support, which approximates its position. The localization integral can be evaluated explicitly. If one of the following holds, then $\sqrt{a}\int \bar{w}(at)w(t-b)\,dt = 0$:

1. if $b \le -1$ or $b \ge \frac{1}{a}$, for fixed a,
2. if $a \le \frac{1}{2}$ or $a \ge 2$, for $b = 0$,
3. if $a \ge \frac{1}{b}$ or $a \le \frac{1}{2(b+1)}$, for fixed $b > 0$,
4. if $a > 0$, for fixed $b \le -1$,
5. if $a \ge \frac{1}{b+1}$, for fixed $b \in (-1, -\frac{1}{2}]$,
6. if $a \ge \frac{2}{2b+1}$, for fixed $b \in (-\frac{1}{2}, 0)$.

The only remaining case, fixed $b \in (-1, 0)$ and $a \to 0+$, gives

$$\sqrt{a}\int \bar{w}(at)w(t-b)\,dt = \begin{cases} -(b+1)\sqrt{a}, & \text{if } -1 < b \le -\frac{1}{2}; \\ b\sqrt{a}, & \text{if } -\frac{1}{2} < b < 0, \end{cases} \tag{5.3}$$

whenever $0 < a < \frac{1}{2(b+1)}$, which means that the integral is $O(\sqrt{a})$ as $a \to 0+$. Cases 1 through 5 are depicted in Figure 5.1. Case 6 and Equation 5.3 are left as exercises.

To show that the Haar wavelet has the completeness property, it is enough to show that all piecewise constant functions are superpositions of Haar functions w_{ab}. But $\mathbf{1}$, the indicator function of $[0, 1)$, is such a superposition, though it takes infinitely many terms. The proof starts by considering the partial sums, namely

$$\phi^J(t) \stackrel{\text{def}}{=} \sum_{j=1}^{J} 2^{-j/2} w_{2^{-j},0}(t) = \sum_{j=1}^{J} \frac{w(t/2^j)}{2^j} = \begin{cases} 0, & \text{if } t < 0; \\ 1 - 2^{-J}, & \text{if } 0 \le t < 1; \\ -2^{-J}, & \text{if } 1 \le t < 2^J; \\ 0, & \text{if } t \ge 2^J. \end{cases} \tag{5.4}$$

The first four summands $\frac{1}{2}w(\frac{1}{2}t), \ldots, \frac{1}{16}w(\frac{1}{16}t)$ are shown together in Figure 5.2, and Figure 5.3 shows the first four partial sums $\phi^1, \ldots, \phi^4$ plotted together.

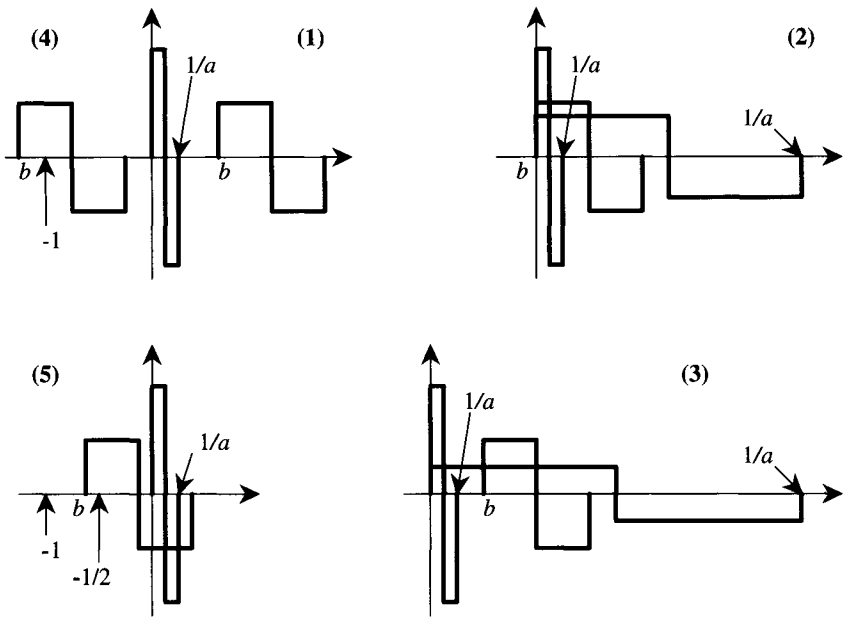


Figure 5.1: Graphs of Haar wavelets $w(at)$ and $w(t-b)$ in most of the listed configurations.

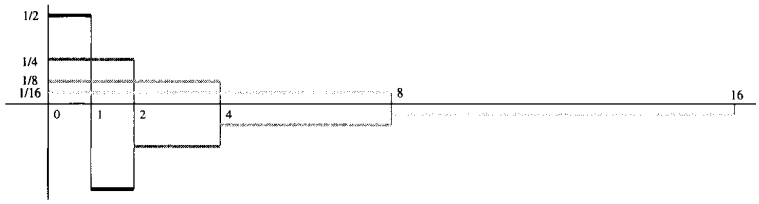


Figure 5.2: Four dilated Haar wavelets.

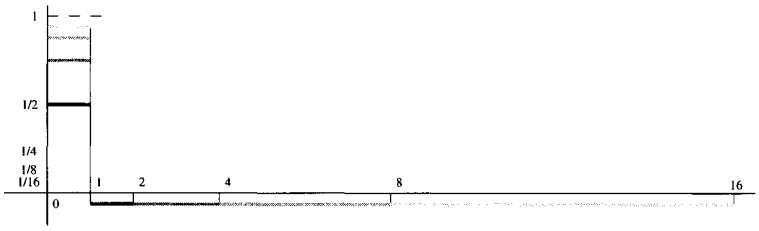


Figure 5.3: Partial sum of four dilated Haar wavelets.

It is clear from the formula that at every point $t \in \mathbf{R}$, $\phi^J(t) \to \mathbf{1}(t)$ as $J \to \infty$. What is more important is that inner products with ϕ^J converge to averages as $J \to \infty$:

Lemma 5.1 *If* $\int_0^\infty |u(t)|^2\,dt$ *exists and is finite, then* $\lim_{J\to\infty} \langle \phi^J, u\rangle = \langle \mathbf{1}, u\rangle$.

Proof: Equation 5.4 implies that

$$\lim_{J\to\infty} \langle \phi^J, u\rangle = \int_0^1 u(t)\,dt - \lim_{J\to\infty} \left[2^{-J}\int_0^{2^J} u(t)\,dt\right]$$

The second term vanishes in the limit by the following estimate: for each J, note that $\mathbf{1}(2^{-J}t)$ is the indicator function of $[0, 2^J)$, and write

$$2^{-J}\int_0^{2^J} u(t)\,dt = 2^{-J}\int_0^\infty \mathbf{1}(2^{-J}t)\,u(t)\,dt.$$

Then apply the Cauchy-Schwarz inequality to the integral:

$$\begin{aligned}\left|\int_0^\infty \mathbf{1}(2^{-J}t)\,u(t)\,dt\right| &\le \left(\int_0^\infty |\mathbf{1}(2^{-J}t)|^2\,dt\right)^{1/2}\left(\int_0^\infty |u(t)|^2\,dt\right)^{1/2} \\ &= 2^{J/2}\left(\int_0^\infty |u(t)|^2\,dt\right)^{1/2}.\end{aligned}$$

This becomes $2^{-J/2}\left(\int_0^\infty |u(t)|^2\,dt\right)^{1/2}$ when multiplied by 2^{-J}. But the integral of $|u|^2$ is finite by assumption, and $2^{-J/2} \to 0$ as $J \to +\infty$, finishing the proof. □

5.1.2 The affine group

The mapping $x \mapsto ax + b$ of a point $x \in \mathbf{R}$, determined by parameters $a > 0$ and $b \in \mathbf{R}$, is called an *affine transformation*. The set of ordered parameter pairs $\{(a, b) : a > 0, b \in \mathbf{R}\}$ with the rules

$$(a', b')(a, b) = (a'a, a'b + b'); \qquad (a, b)^{-1} = (a^{-1}, -a^{-1}b), \tag{5.5}$$

is called the *affine group* or the *Ax+B group*, and is denoted by **Aff**. The affine group's identity element is $(1, 0)$, denoted by e.

The multiplication rule comes from composition:

$$x \mapsto a'(ax + b) + b' = a'ax + a'b + b'. \tag{5.6}$$

Note that the left factor (a', b') is applied second, much as matrix A' is applied to vector x second in the composition $A'Ax$. A *matrix representation* of **Aff** is a function ρ from group elements (a, b) to invertible matrices $\rho(a, b)$, which converts group multiplication into matrix multiplication. One such representation is

$$(a, b) \mapsto \rho(a, b) = \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix}. \tag{5.7}$$

Matrix $\rho(a,b)$ sends vector $\begin{pmatrix} x \\ 1 \end{pmatrix}$ to vector $\begin{pmatrix} ax+b \\ 1 \end{pmatrix}$. It is left as an exercise to check that $\rho((a',b')(a,b)) = \rho(a',b')\rho(a,b)$.

There are many other matrix representations, including the obvious generalization

$$(a,b) \mapsto \begin{pmatrix} a & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.8}$$

The constant function $\rho(a,b) = 1$ is also a matrix representation of **Aff**, since it preserves the multiplication, but it is not *faithful*, or one-to-one, because it sends many different group elements to the same 1×1 identity matrix.

Unitary representations

We can also interpret $(a,b) \in \mathbf{Aff}$ as the substitution $t \leftarrow (t-b)/a$ that sends a function f to f_{ab}. This gives a more general *representation* of **Aff** as linear transformations, which are generalizations of matrices. For each $(a,b) \in \mathbf{Aff}$, define

$$\sigma(a,b)f(t) = f_{ab}(t) \stackrel{\text{def}}{=} \frac{1}{\sqrt{a}} f\left(\frac{t-b}{a}\right). \tag{5.9}$$

This $\sigma(a,b)$ is a linear transformation, because addition in any function space occurs pointwise. Namely, for any functions f, g and any scalar c,

$$\begin{aligned} \sigma(a,b)[f+cg](t) &= \frac{1}{\sqrt{a}}[f+cg]\left(\frac{t-b}{a}\right) = \frac{1}{\sqrt{a}} f\left(\frac{t-b}{a}\right) + \frac{1}{\sqrt{a}} cg\left(\frac{t-b}{a}\right) \\ &= \sigma(a,b)f(t) + c\sigma(a,b)g(t). \end{aligned}$$

In fact, $\sigma(a,b)$ is an invertible linear transformation, with inverse $\sigma(a,b)^{-1}f(t) = \sqrt{a}f(at+b)$, and the inverse $\sigma(a,b)^{-1}$ represents $(a,b)^{-1} = (a^{-1}, -ba^{-1})$.

Composing such substitutions shows how the group multiplication is converted into composition of linear transformations:

$$\sigma(a',b')\sigma(a,b)f(t) = \sigma(a',b')g(t) = \frac{1}{\sqrt{a'}} g\left(\frac{t-b'}{a'}\right),$$

where $g(s) = \sigma(a,b)f(s) = \frac{1}{\sqrt{a}} f\left(\frac{s-b}{a}\right)$. But then, putting $s = \frac{t-b'}{a'}$ gives

$$\begin{aligned} \sigma(a',b')\sigma(a,b)f(t) &= \frac{1}{\sqrt{a'}}\frac{1}{\sqrt{a}} f\left(\frac{\frac{t-b'}{a'} - b}{a}\right) = \frac{1}{\sqrt{a'a}} f\left(\frac{t-(a'b+b')}{a'a}\right) \\ &= \sigma(a'a, a'b+b')f(t). \end{aligned}$$

Observe that σ is faithful, if the space has enough functions: $\sigma(a,b) = \sigma(a',b')$ means that $\sigma(a,b)f(t) = \sigma(a',b')f(t)$ for every function f and every t. There must simply be enough nonconstant functions so that this can only hold for all t if $a = a'$ and $b = b'$.

For definiteness, suppose now that $f = f(t)$ belongs to $L^2(\mathbf{R})$, the inner product space of complex-valued, square-integrable functions of one real variable, with inner product $\langle u, v \rangle = \int_{-\infty}^{\infty} \bar{u}(t)v(t)\,dt$. Then $\sigma(a,b) : L^2(\mathbf{R}) \to L^2(\mathbf{R})$, and the factor $1/\sqrt{a}$ insures that $\|\sigma(a,b)f\| = \|f\|$. This space contains the Haar wavelet, and is a natural place to look for other wavelets. It is big enough to make σ faithful, because it contains $\mathbf{1}$: if $\sigma(a,b) = \sigma(a',b')$, then

$$\mathbf{1}(t) = \sigma(a,b)^{-1}\sigma(a',b')\mathbf{1}(t) = \mathbf{1}\left(\frac{at+b-b'}{a'}\right) = \mathbf{1}\left(\frac{a}{a'}t + \frac{b-b'}{a'}\right).$$

This can only hold for all t if $a' = a$ and $b' = b$.

Recall that a linear transformation $U : \mathbf{X} \to \mathbf{X}$ on an inner product space $\mathbf{X}$ is called *unitary* if and only if, for all $f, g \in \mathbf{X}$, we have:

$$\langle Ug, Uf \rangle = \langle g, f \rangle . \tag{5.10}$$

Unitary transformations generalize rotations of geometric objects about the origin. They are rigid motions that preserve lengths as well as angles:

$$\|Uf\| = \sqrt{\langle Uf, Uf \rangle} = \sqrt{\langle f, f \rangle} = \|f\|. \tag{5.11}$$

But for each $(a,b) \in \mathbf{Aff}$, the linear transformation $\sigma(a,b)$ is unitary:

$$\begin{aligned} \langle \sigma(a,b)g, \sigma(a,b)f \rangle &= \int_{-\infty}^{\infty} \frac{1}{\sqrt{a}}\bar{g}\left(\frac{t-b}{a}\right)\frac{1}{\sqrt{a}}f\left(\frac{t-b}{a}\right)dt \\ &= \int_{-\infty}^{\infty} \bar{g}(t)f(t)\,dt \quad = \langle g, f \rangle , \end{aligned} \tag{5.12}$$

after the substitution $t \leftarrow at + b$.

Some other examples of unitary transformations on $L^2(\mathbf{R})$ are given in Table 5.1. To show these are unitary, perform the appropriate changes of variable in the integrals behind Equation 5.10.

The transformation $f \mapsto f_{ab}$ is a combination of dilation followed by translation: $f_{ab} = \tau_b \delta_a f$. Thus $\sigma(a,b) = \tau_b \delta_a$. This gives another proof that $\sigma(a,b)$ is unitary, since the product UV of unitary transformations U and V is unitary:

$$\langle UVg, UVf \rangle = \langle Vg, Vf \rangle = \langle g, f \rangle , \qquad \text{for all } f, g.$$

Recall that the Fourier integral transform is defined by the following formula:

$$\mathcal{F}u(\xi) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} e^{-2\pi i t\xi} u(t)\,dt. \tag{5.13}$$

$\mathcal{F}$ is itself a unitary transformation because of Plancherel's theorem, Theorem 3.11. But it has a special relationship with these other unitary transformations. The

Name	Action on $f = f(t)$	Inverse
reflection	$\nu f(t) \stackrel{\text{def}}{=} f(-t)$	$\nu^{-1} = \nu$
dilation	$\delta_a f(t) \stackrel{\text{def}}{=} \frac{1}{\sqrt{a}} f(\frac{t}{a})$	$\delta_a^{-1} = \delta_{(1/a)}$
translation	$\tau_b f(t) \stackrel{\text{def}}{=} f(t-b)$	$\tau_b^{-1} = \tau_{(-b)}$
modulation	$\mu_c f(t) \stackrel{\text{def}}{=} e^{2\pi i c t} f(t)$	$\mu_c^{-1} = \mu_{(-c)}$

Table 5.1: Various unitary linear transformations of functions.

Fourier integral transform of w_{ab} can be expressed in terms of the Fourier integral transform of w:

$$\begin{aligned} \mathcal{F}w_{ab}(\xi) &= \int_{-\infty}^{\infty} e^{-2\pi i t\xi} \frac{1}{\sqrt{a}} w(\frac{t-b}{a})\, dt = \int_{-\infty}^{\infty} e^{-2\pi i(at'+b)\xi} \sqrt{a}\, w(t')\, dt' \\ &= \sqrt{a}\, e^{-2\pi i b\xi} \mathcal{F}w(a\xi). \end{aligned} \tag{5.14}$$

Another way of saying this is that $\mathcal{F}w_{ab} = \mathcal{F}\tau_b\delta_a w = \mu_{-b}\delta_{(1/a)}\mathcal{F}w$, or that after Fourier transformation, translation becomes inverse modulation, and dilation becomes inverse dilation:

$$\mathcal{F}\tau_b\mathcal{F}^* = \mu_b^{-1}; \qquad \mathcal{F}\delta_a\mathcal{F}^* = \delta_a^{-1}.$$

Likewise, after Fourier transformation, reflection becomes inverse reflection, which is again reflection:

$$\begin{aligned} \mathcal{F}(\nu w)(\xi) &= \int_{-\infty}^{\infty} e^{-2\pi i t\xi} w(-t)\, dt = \int_{-\infty}^{\infty} e^{-2\pi i(-t)\xi} w(t)\, dt \\ &= \int_{-\infty}^{\infty} e^{-2\pi i t(-\xi)} w(t)\, dt = \mathcal{F}w(-\xi). \end{aligned} \tag{5.15}$$

Thus $\mathcal{F}\nu\mathcal{F}^* = \nu^{-1} = \nu$.

A *unitary representation* of the affine group is a faithful representation of group elements as unitary linear transformations. Equation 5.12 shows that $(a,b) \mapsto \sigma(a,b)$ is a unitary representation. The faithful matrix representation $(a,b) \mapsto \rho(a,b)$ of Equation 5.7, however, is not unitary.

Integration on the group

Suppose that $f = f(a,b)$ is a function defined on two real variables, $a > 0$ and b. Then f may be regarded as a function on **Aff**. It is possible to define an area

element $d\mathbf{a}$ for $\mathbf{a} = (a, b) \in \mathbf{Aff}$ so that, for any fixed $\mathbf{a}' \in \mathbf{Aff}$,

$$\int_{\mathbf{Aff}} f(\mathbf{a})\, d\mathbf{a} = \int_{\mathbf{Aff}} f(\mathbf{a}'\mathbf{a})\, d\mathbf{a}. \tag{5.16}$$

In other words, the integral should not be changed by applying $\mathbf{a}'$ and thus renaming the group elements. Such *left invariance* requires that $|d(\mathbf{a}'\mathbf{a})/d\mathbf{a}| = 1$, namely, the Jacobian of the change of variable $\mathbf{a} \leftarrow \mathbf{a}'\mathbf{a}$ is the constant 1. An area element $d\mathbf{a}$ with these properties is called a *left invariant measure*.

Now **Aff** is parameterized by (a, b), so $d\mathbf{a} = h(\mathbf{a})\, da db = h(a, b)\, da db$, where h is an unknown function to be determined. The invariance condition implies

$$\begin{aligned} 1 &= \left|\frac{d(\mathbf{a}'\mathbf{a})}{d\mathbf{a}}\right| = \frac{h(\mathbf{a}'\mathbf{a})}{h(\mathbf{a})}\left|\frac{\partial(a'', b'')}{\partial(a, b)}\right| = \frac{h(\mathbf{a}'\mathbf{a})}{h(\mathbf{a})}\begin{vmatrix} da''/da & da''/db \\ db''/da & db''/db \end{vmatrix} \\ &= \frac{h(\mathbf{a}'\mathbf{a})}{h(\mathbf{a})}\begin{vmatrix} a' & 0 \\ 0 & a' \end{vmatrix} = \frac{h(\mathbf{a}'\mathbf{a})}{h(\mathbf{a})}\,|a'|^2. \end{aligned} \tag{5.17}$$

Here $\mathbf{a}'\mathbf{a} = (a', b')(a, b) = (a'', b'')$ defines the coordinate functions $a'' = a'a$, $b'' = a'b + b'$ from Equation 5.5. Thus $h(\mathbf{a}'\mathbf{a}) = \frac{1}{|a'|^2} h(\mathbf{a})$, so putting $\mathbf{a} = (1, 0)$ to force $\mathbf{a}'\mathbf{a} = \mathbf{a}'$ yields

$$h(a, b) = h(1, 0)\frac{1}{a^2}. \tag{5.18}$$

Thus, the *left invariant integral* of f over **Aff** is uniquely determined up to the arbitrary constant $h(1, 0)$. Choosing the normalization $h(1, 0) = 1$ gives normalized left invariant measure, or *Haar measure*, on **Aff**, and uniquely defines the normalized left invariant integral:

$$\int_{\mathbf{Aff}} f(\mathbf{a})\, d\mathbf{a} \stackrel{\text{def}}{=} \int_{b=-\infty}^{\infty}\int_{a=0}^{\infty} f(a, b)\,\frac{da db}{a^2}. \tag{5.19}$$

What functions f are integrable on **Aff**? First, because the domain of integration is unbounded, it is necessary that $f(a, b) \to 0$ sufficiently fast as $b \to \pm\infty$ and as $a \to +\infty$. But also, because there is a factor $1/a^2$ in the invariant measure, it is necessary that $f(a, b)/a^2$ be integrable in a as $a \to 0+$, which means $f(a, b) \to 0$ somewhat faster than a as $a \to 0$. An example of such a function is $f(a, b) = a^2 e^{-a-|b|}$, for which

$$\begin{aligned} \int_{\mathbf{Aff}} f(\mathbf{a})\, d\mathbf{a} &= \int_{-\infty}^{\infty}\int_{0}^{\infty} a^2 e^{-a-|b|}\,\frac{da db}{a^2} = \int_{-\infty}^{\infty}\int_{0}^{\infty} e^{-a-|b|}\, da db \\ &= \left(\int_{-\infty}^{\infty} e^{-|b|}\, db\right)\left(\int_{0}^{\infty} e^{-a}\, da\right) = 2. \end{aligned}$$

5.1.3 Wavelet transforms

Fix a wavelet w and define the associated *wavelet transform* Wu of any function $u = u(t)$ in $L^2(\mathbf{R})$ by the formula

$$Wu : \mathbf{Aff} \to \mathbf{C}; \qquad Wu(a, b) \stackrel{\text{def}}{=} \langle w_{ab}, u\rangle = \int_{-\infty}^{\infty} \frac{1}{\sqrt{a}}\,\bar{w}\left(\frac{t - b}{a}\right) u(t)\, dt. \tag{5.20}$$

For w in $L^2(\mathbf{R})$, this formula makes sense for any u in $L^2(\mathbf{R})$. The numbers $\{Wu(a,b) : (a,b) \in \mathbf{Aff}\}$ are the amplitudes of the function u decomposed into τ_b-translated and δ_a-dilated versions of the mother function w. It is roughly analogous to the Fourier integral transform amplitudes, $\{\mathcal{F}u(\xi) : \xi \in \mathbf{R}\}$, of the signal u decomposed into μ_ξ-modulated versions $e^{2\pi i\xi t}$ of the constant function 1. The set $\mathbf{R}$, which is a group under addition, plays the role of **Aff** for $\mathcal{F}$. However, since $\mu_\xi 1(x) = e^{2\pi i x\xi}$ does not belong to $L^2(\mathbf{R})$ for any ξ, there is no analogous expression $\mathcal{F}u(\xi) = \langle \mu_\xi 1, u\rangle$.

A wavelet transform of a function is indexed by the parameters (a,b) of the affine group, just as the Fourier transform is indexed by the frequency parameter. Unlike the Fourier transform, there is no single wavelet transform; it varies with the choice of mother function w.

The wavelet transform of a function is as well-behaved as the wavelet w. For example, since $|\langle w_{ab}, u\rangle| \le \|w_{ab}\|\,\|u\| = \|w\|\,\|u\|$, this implies that $|Wu(a,b)| \le \|w\|\,\|u\|$ for all $(a,b) \in \mathbf{Aff}$. Thus the wavelet transform of a square-integrable function, using a square-integrable mother function w, is a bounded function on **Aff**. Likewise, if w is smooth and has compact support, then Wu is a smooth function of a and b. This is proved by differentiating with respect to a or b under the integral sign in Equation 5.20.

For what w can u be recovered from Wu? Clearly $w = 0$ won't work, so that trivial case is excluded by the normalization assumption $\|w\| = 1$.

For certain special w, there is a formula that recovers averages of a function from its wavelet transform. For example, suppose w is the Haar mother function defined in Equation 5.2. Since $2^{-j/2}Wu(2^j,0)$ is the inner product of u with $\frac{1}{2^j}w(\frac{1}{2^j}t)$, the sum of certain samples of Wu is the inner product of u with the corresponding sum of dilated w's:

$$\sum_{j=1}^{J} 2^{-j/2}Wu(2^j,0) = \langle \phi^J, u\rangle \to \langle \mathbf{1}, u\rangle\,, \qquad \text{as } J \to \infty, \tag{5.21}$$

where $\phi^J(t) = \sum_{j=1}^{J} \frac{1}{2^j}w(\frac{1}{2^j}t)$ as in Lemma 5.1. To get the value of u at a point t, just take averages of u over the interval $[t, t+2^{-M}]$ by starting the sum at $j = -M$ and let $M \to \infty$. For example, if u is continuous at $t = 0$, then

$$\sum_{j=-\infty}^{\infty} 2^{-j/2}Wu(2^j,0) \overset{\text{def}}{=} \lim_{M\to\infty} \sum_{j=-M}^{\infty} 2^{-j/2}Wu(2^j,0) = u(0). \tag{5.22}$$

A necessary condition for invertibility is that $u \mapsto Wu$ is one-to-one. Namely, if $u \ne v$, then $Wu \ne Wv$. With nice wavelets for which Wu is a continuous function of (a,b), this is equivalent to the property that $Wu(a,b) \ne Wv(a,b)$ for some $(a,b) \in \mathbf{Aff}$, if $u \ne v$. But since $Wu - Wv = \langle w_{ab}, u\rangle - \langle w_{ab}, v\rangle = \langle w_{ab}, u - v\rangle = W[u-v]$, this is equivalent to the condition that if $u \ne 0$, then $\langle w_{ab}, u\rangle \ne 0$ for some $(a,b) \in \mathbf{Aff}$.

Another way that W is one-to-one is that $\|Wu\|_{\mathbf{Aff}} = 0$ if and only if $\|u\| = 0$. The Hermitean inner product on **Aff**, and thus the norm $\|\cdot\|_{\mathbf{Aff}}$, is defined using

the left invariant integral on the group:

$$\langle u, v\rangle_{\mathbf{Aff}} \stackrel{\text{def}}{=} \int_{\mathbf{Aff}} \bar{u}(\mathbf{a})v(\mathbf{a})\, d\mathbf{a} = \int_{b=-\infty}^{\infty}\int_{a=0}^{\infty} \bar{u}(a,b)v(a,b)\,\frac{dadb}{a^2} \tag{5.23}$$

This kind of one-to-one condition follows from the analog of the Plancherel theorem, for the wavelet transform rather than the Fourier transform:

Theorem 5.2 *Suppose that the wavelet w is square-integrable and there is a finite, nonzero constant c_w with*

$$c_w = \int_{\xi=0}^{\infty} \frac{|\mathcal{F}w(\xi)|^2}{\xi}\, d\xi = \int_{\xi=0}^{\infty} \frac{|\mathcal{F}w(-\xi)|^2}{\xi}\, d\xi.$$

Then for all square-integrable $u = u(t)$ and $v = v(t)$, we have $\langle Wu, Wv\rangle_{\mathbf{Aff}} = c_w \langle u, v\rangle$.

Proof: Evaluate the inner product $\langle Wu, Wv\rangle_{\mathbf{Aff}}$:

$$\int_{b=-\infty}^{\infty}\int_{a=0}^{\infty} \overline{\left[\int_{t=-\infty}^{\infty} \bar{w}_{ab}(t)u(t)\,dt\right]}\left[\int_{s=-\infty}^{\infty} \bar{w}_{ab}(s)v(s)\,ds\right]\frac{dadb}{a^2} =$$

$$= \int_{s,t=-\infty}^{\infty} \bar{u}(t)v(s)\int_{a=0}^{\infty}\left[\int_{b=-\infty}^{\infty} \frac{1}{\sqrt{a}}\bar{w}\left(\frac{s-b}{a}\right)\frac{1}{\sqrt{a}}w\left(\frac{t-b}{a}\right)db\right]\frac{da}{a^2}\,dsdt.$$

The innermost integral, the one with respect to the parameter b, is $\langle w^s, w^t\rangle$, where $w^t = w^t(b) \stackrel{\text{def}}{=} \frac{1}{\sqrt{a}}w(\frac{t-b}{a})$ and $w^s = w^s(b) \stackrel{\text{def}}{=} \frac{1}{\sqrt{a}}w(\frac{s-b}{a})$ are considered functions of b. By Plancherel's theorem, $\langle w^s, w^t\rangle = \langle \mathcal{F}w^s, \mathcal{F}w^t\rangle$, and a direct computation yields

$$\mathcal{F}w^t(\beta) = \sqrt{a}e^{2\pi i t\beta}\mathcal{F}w(-a\beta); \qquad \mathcal{F}w^s(\beta) = \sqrt{a}e^{2\pi i s\beta}\mathcal{F}w(-a\beta).$$

Thus,

$$\langle w^s, w^t\rangle = a\int_{\beta=-\infty}^{\infty} e^{2\pi i(t-s)\beta}\,|\mathcal{F}w(-a\beta)|^2\,d\beta = a\int_{\beta=-\infty}^{\infty} e^{2\pi i(s-t)\beta}\,|\mathcal{F}w(a\beta)|^2\,d\beta,$$

and so, after substitution and interchanging the a and β integration,

$$\langle Wu, Wv\rangle_{\mathbf{Aff}} = \int_{s,t=-\infty}^{\infty} \bar{u}(t)v(s)\int_{\beta=-\infty}^{\infty} e^{2\pi i(s-t)\beta}\left(\int_{a=0}^{\infty}|\mathcal{F}w(a\beta)|^2\,\frac{da}{a}\right)d\beta\,dsdt.$$

Now observe that for $\beta > 0$,

$$\int_0^{\infty} f(a\beta)\,\frac{da}{a} = \int_0^{\infty} f(a')\,\frac{da'}{a'}, \tag{5.24}$$

if both integrals converge, which is the case if, for example, $f(x) \to 0$ sufficiently rapidly as $x \to 0+$ and as $x \to \infty$. Equality is proved by substituting $a \leftarrow a'/\beta$. For $\beta < 0$, the substitution $a \leftarrow -a''/\beta$ yields

$$\int_0^\infty f(a\beta)\,\frac{da}{a} = \int_0^\infty f(-a'')\,\frac{da''}{a''}. \tag{5.25}$$

Therefore, if c_w is finite and nonzero,

$$\langle Wu, Wv\rangle_{\mathbf{Aff}} = c_w \int_{\beta=-\infty}^{\infty}\int_{t=-\infty}^{\infty}\int_{s=-\infty}^{\infty} e^{2\pi i(s-t)\beta}\bar{u}(t)v(s)\,dsdtd\beta,$$

and the result follows from Corollary 3.12. □

Any square-integrable function w for which c_w is finite and nonzero is called *admissible*, and the resulting c_w is called its *normalization constant*. Such functions give invertible wavelet transforms: $c_w\,\langle u, v\rangle = \langle Wu, Wv\rangle_{\mathbf{Aff}} = \langle u, W^*Wv\rangle$ for all $u, v \in L^2(\mathbf{R})$, so $\frac{1}{c_w}W^*Wv = v$ and $\frac{1}{c_w}W^*W = Id$. Thus the *inverse wavelet transform* is $W^{-1} = \frac{1}{c_w}W^*$, where W^* is a linear transformation defined for functions on **Aff** by

$$W^*z(t) = \langle \bar{w}_{ab}(t), z(a,b)\rangle_{\mathbf{Aff}} = \int_{b=-\infty}^{\infty}\int_{a=0}^{\infty} \frac{1}{\sqrt{a}}w\left(\frac{t-b}{a}\right) z(a,b)\,\frac{dadb}{a^2}. \tag{5.26}$$

The adjoint wavelet transform W^* is derived by changing the order of integration: for any $u = u(t)$,

$$\begin{aligned}
&\langle Wu(a,b), z(a,b)\rangle_{\mathbf{Aff}} \;= \\
&= \int_{b=-\infty}^{\infty}\int_{a=0}^{\infty} \overline{Wu(a,b)}\, z(a,b)\,\frac{dadb}{a^2} \\
&= \int_{b=-\infty}^{\infty}\int_{a=0}^{\infty} \overline{\left[\int_{t=-\infty}^{\infty} \frac{1}{\sqrt{a}}\bar{w}\left(\frac{t-b}{a}\right) u(t)\,dt\right]}\, z(a,b)\,\frac{dadb}{a^2} \\
&= \int_{t=-\infty}^{\infty} \bar{u}(t)\left[\int_{b=-\infty}^{\infty}\int_{a=0}^{\infty} \frac{1}{\sqrt{a}}w\left(\frac{t-b}{a}\right) z(a,b)\,\frac{dadb}{a^2}\right] dt \;=\; \langle u, W^*z\rangle\,.
\end{aligned}$$

Since this holds for all $u \in L^2(\mathbf{R})$, the nondegeneracy of the inner product allows us to identify W^*z with the formula inside the last set of square brackets, which is evidently $\langle \bar{w}_{ab}(t), z(a,b)\rangle_{\mathbf{Aff}}$.

Theorem 5.3 (Wavelet Inversion) *Suppose that $w = w(t)$ is admissible with normalization constant $c_w \neq 0$. Then its associated wavelet transform satisfies $W : L^2(\mathbf{R}) \to L^2(\mathbf{Aff})$, its adjoint satisfies $W^* : L^2(\mathbf{Aff}) \to L^2(\mathbf{R})$, and we have*

1. $[\frac{1}{c_w}W^*W]u = u$, *for any function $u = u(t) \in L^2(\mathbf{R})$;*
2. $[\frac{1}{c_w}WW^*]z = z$, *for any function $z = z(a,b) \in L^2(\mathbf{Aff})$.*

Proof: Theorem 5.2 with $u = v \in L^2(\mathbf{R})$ implies that $Wu \in L^2(\mathbf{Aff})$, with $\|Wu\|_a \leq c_w \|u\|_r$. The norms $\|\cdot\|_a$ and $\|\cdot\|_r$ are derived from the inner products in $L^2(\mathbf{Aff})$ and $L^2(\mathbf{R})$, respectively Thus, fixing $z \in L^2(\mathbf{Aff})$ and taking any unit vector $u \in L^2(\mathbf{R})$ gives

$$|\langle u, W^* z \rangle| = |\langle Wu, z \rangle_{\mathbf{Aff}}| \leq \|Wu\|_a \|z\|_a = c_w \|z\|_a,$$

by the Cauchy–Schwarz inequality in $L^2(\mathbf{Aff})$. Taking the supremum over all such unit vectors gives the estimate $\|W^* z\|_r \leq c_w \|z\|_a$, so $W^* z \in L^2(\mathbf{R})$. The inversion results now follow from the discussion preceding Equation 5.26. □

Because the admissibility condition applies to $\mathcal{F}w$, wavelets are often defined as inverse Fourier integral transforms of bump functions with simple formulas. For example, we may define $w = w(t)$ by $\mathcal{F}w(\xi) = \phi(|\xi|)$, where

$$\phi(\xi) = \begin{cases} e^{-(\log \xi)^2}, & \text{if } \xi > 0; \\ 0, & \text{if } \xi \leq 0. \end{cases} \tag{5.27}$$

Then $\mathcal{F}w$ is a pair of bumps peaking at frequencies $\xi = \pm 1$. It is an exercise to show that $w = w(t)$ is a smooth, rapidly decaying function concentrated near $t = 0$, so that w is localized in both time and frequency. Since w is Hermitean symmetric in frequency, it is real-valued. But also, w is an admissible mother function. The integrands are positive so the admissibility integrals are positive:

$$0 < \int_0^\infty \frac{|\mathcal{F}w(\xi)|^2}{\xi}\, d\xi = \int_0^\infty \frac{|\mathcal{F}w(-\xi)|^2}{\xi}\, d\xi < \infty.$$

Finiteness follows from the rapid decay of $\phi(\xi)$ as $\xi \to \infty$ and as $\xi \to 0+$. Showing this rapid decay is left as an exercise.

The Fourier integral transform of the dilation, $\mathcal{F}w_{a0}$, has peaks at $\xi = \pm 1/a$. The location of these frequency peaks is the same for $w_{ab}(t)$, which is concentrated near $t = b$. Hence, by Plancherel's theorem, $Wu(a, b) = \langle w_{ab}, u \rangle = \langle \mathcal{F}w_{ab}, \mathcal{F}u \rangle$ is an average of u's frequency components near $\xi = \pm 1/a$ and $t = b$.

Any admissible wavelet must have a power spectrum with energy at both positive and negative frequencies. If we are not interested in an invertible transform, but only wish to identify time-frequency components of a signal, then we can use just one of the two halves of the admissible function above. For example, putting $w^+ \stackrel{\text{def}}{=} \mathcal{F}^{-1}\phi$ gives a complex-valued wavelet transform, and if $|W^+ f(a, b)| = |\langle w^+_{ab}, f \rangle|$ is large, then we may conclude that near time b the signal f has a strong component with frequencies near $1/a$.

5.2 Discrete Wavelet Transforms

Whenever ψ is an admissible mother function, the wavelet inversion theorem implies that a function u can be recovered from $\{Wu(a, b) : a > 0, b \in \mathbf{R}\}$, its wavelet transform. But samples of Wu at discrete values of (a, b) also determine u, and

good approximations to u may be recovered from finite lists of these samples. Furthermore, only certain averages of u are needed to compute the samples of Wu. The relation between discrete averages of u and samples of Wu is called the *discrete wavelet transform*, or DWT.

To get a discrete approximate for $u \in L^2(\mathbf{R})$, we first fix a *scaling function* $\phi \in L^2$ such that $B = \{\phi_n(t) = \phi(t-n) : n \in \mathbf{Z}\}$, is an orthonormal set. Then $V \stackrel{\text{def}}{=} \overline{\text{span}}\, \mathbf{B}$, which is called a *scale space*, has B as an orthonormal Schauder basis, and the following transformation is an orthogonal projection onto V:

$$P : L^2(\mathbf{R}) \to V; \qquad Pu(t) \stackrel{\text{def}}{=} \sum_{n \in \mathbf{Z}} \langle \phi_n, u \rangle \, \phi_n(t). \tag{5.28}$$

This is the same formula as Equation 4.22, only V is not necessarily a sampling space since ϕ is not necessarily an interpolating function. The sequence of expansion coefficients, $\{\langle \phi_n, u \rangle : n \in \mathbf{Z}\}$, is the discrete approximation to u.

For example, we may approximate u by a discrete sequence of local average values. Let $\mathbf{1} = \mathbf{1}(t)$ be the indicator function of the interval $[0,1)$. Put $\phi_n = \phi_n(t) \stackrel{\text{def}}{=} \mathbf{1}(t-n)$; then ϕ_n is the indicator function of the interval $[n, n+1)$, and it is easy to see that $\{\phi_n : n \in \mathbf{Z}\}$ is an orthonormal set. For any u, the function Pu defined by Equation 4.22 with these ϕ's will be piecewise constant on intervals of the form $[n, n+1)$, where n is an integer. The constant value on $[k, k+1)$ is just an average:

$$Pu(t) = \int_{t=k}^{k+1} u(t)\, dt, \qquad \text{if } t \in [k, k+1).$$

The Haar wavelet transform of such a piecewise constant u is given by

$$Wu(a,b) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right) u(t)\, dt = \frac{1}{\sqrt{a}} \int_{b}^{b+\frac{a}{2}} u(t)\, dt - \frac{1}{\sqrt{a}} \int_{b+\frac{a}{2}}^{b+a} u(t)\, dt. \tag{5.29}$$

Putting $b = k$, an integer, and $a = 2^j$ for integer $j > 0$, this evaluates to

$$\begin{aligned} Wu(2^j, k) \;=\; & \frac{1}{2^{j/2}} \left[u(k) + \cdots + u(k + 2^{j-1} - 1)\right] \\ & - \frac{1}{2^{j/2}} \left[u(k + 2^{j-1}) + \cdots + u(k + 2^j - 1)\right] \end{aligned}$$

In particular, $\sqrt{2}\, Wu(2^1, 0) = u(0) - u(1)$, $\sqrt{2}\, Wu(2^1, 1) = u(1) - u(2)$, and so on.

5.2.1 Multiresolution analysis (MRA)

If $\mathbf{1} = \mathbf{1}(t)$ is the indicator function of $[0,1)$, then $\mathbf{1}(2t)$ is the indicator function of $[0, \frac{1}{2})$ and $\mathbf{1}(2t-1)$ is the indicator function of $[\frac{1}{2}, 1)$. Thus

$$\mathbf{1}(t) = \mathbf{1}(2t) + \mathbf{1}(2t-1). \tag{5.30}$$

This is one example of a *two-scale relation*, because it relates a function to shifts of itself dilated to another scale. Likewise,

$$\psi(t) = \mathbf{1}(2t) - \mathbf{1}(2t-1), \tag{5.31}$$

where ψ is the Haar mother function defined in Equation 5.2. Replacing $t \leftarrow 2^j t - k$ in Equations 5.30 and 5.31 gives

$$\begin{aligned} \mathbf{1}(2^j t - k) &= \mathbf{1}(2^{j+1}t - 2k) + \mathbf{1}(2^{j+1}t - 2k - 1), \\ \psi(2^j t) &= \mathbf{1}(2^{j+1}t - 2k) - \mathbf{1}(2^{j+1}t - 2k - 1). \end{aligned}$$

Any nonzero[1] function ϕ satisfying an equation of the form

$$\phi(t) = \sum_k h(k)\sqrt{2}\,\phi(2t-k) \stackrel{\text{def}}{=} H\phi(t), \tag{5.32}$$

for a possibly infinite sequence $h = \{h(k) : k \in \mathbf{Z}\}$ of *filter coefficients*, is said to satisfy a *two-scale relation*. The sequence[2] h is called a *quadrature filter*, or simply *filter*. The factor $\sqrt{2}$ is used so that we have unit vectors on both sides: $\|\sqrt{2}\phi(2t-k)\| = \|\phi\| = 1$, if $\phi \in L^2(\mathbf{R})$ is a unit vector, for all shifts k.

It will be assumed that $h(k) = 0$ for all but finitely many values of k, as in the Haar case where $h(0) = \frac{1}{\sqrt{2}}$, $h(1) = \frac{1}{\sqrt{2}}$, and $h(k) = 0$ for all $k \neq 0, 1$. Such an h is called a *finite impulse response*, FIR, or simply *finite*, filter. The finiteness guarantees that H makes sense on any vector space.

Substituting $t \leftarrow t - n$ into Equation 5.32, then renumbering $k \leftarrow k' - 2n$, shows that

$$\phi(t-n) = \sum_k h(k)\sqrt{2}\,\phi(2t-2n-k) = \sum_{k'} h(k'-2n)\sqrt{2}\,\phi(2t-k'). \tag{5.33}$$

This means that $\phi(t-n)$ is a finite linear combination of functions in $\{\phi(2t-k) : k \in \mathbf{Z}\}$. Recall that in Equation 2.19, we called the set of all such finite linear combinations a *linear span*. We may thus define a sequence of scale subspaces, beginning with the set of finite linear combinations of elements of $\{\phi(t-k) : k \in \mathbf{Z}\}$:

$$V_0 \stackrel{\text{def}}{=} \text{span}\,\{\phi(t-k) : k \in \mathbf{Z}\}, \tag{5.34}$$

This is the subspace of $L^2(\mathbf{R})$ of all superpositions $\sum_{k\in\mathbf{Z}} s(k)\phi(t-k)$ where $s(k) = 0$ for all but finitely many values of k. Continuing, we define

$$V_j \stackrel{\text{def}}{=} \text{span}\,\{\phi(2^{-j}t - k) : k \in \mathbf{Z}\}, \tag{5.35}$$

the linear span of translates by $2^j k$ of the dilate of ϕ by 2^{-j}.

Functions in V_0 are superpositions of *unit scale* building blocks $\phi(t-k)$; functions in V_j are superpositions of building blocks of scale 2^j. Note that $u(t)$ belongs to V_j if and only if $u(2^j t)$ belongs to V_0; this is a special case of the following lemma:

[1] The uninteresting example $\phi = 0$ obviously satisfies such a relation; that is why it is excluded.

[2] The linear transformation H determined by the sequence is also called a filter.

Lemma 5.4 *For any function $u = u(t)$, $u(t)$ belongs to V_q if and only if $u(2^p t)$ belongs to V_{q-p}.*

Proof: Writing $u(t) = \sum_k s(k)\phi(2^{-q}t - k)$, substitute $t \leftarrow 2^p t$ to get $u(2^p t) = \sum_k s(k)\phi(2^{p-q}t - k)$. □

The special case is obtained when $p = q = j$.

Lemma 5.4 and Equation 5.33 imply that the spaces V_j are *nested*, or *ordered by inclusion*:

Lemma 5.5 *For each j, $V_j \subset V_{j-1}$. That is, if $u \in V_j$, then $u \in V_{j-1}$.*

Proof: First show that $V_0 \subset V_{-1}$. For $u(t) = \sum_n s(n)\phi(t-n) \in V_0$, Equation 5.33 implies

$$u(t) = \sum_n s(n)\phi(t-n) = \sum_n \sum_k s(n)h(k-2n)\phi(2t-k).$$

Thus $u(t) = \sum_k d(k)\phi(2t-k) \in V_{-1}$, where $d(k) = \sum_n s(n)h(k-2n)$.

Now take any $u \in V_j$. By Lemma 5.4 with $p = q = j$, $u(2^j t) \in V_0$, so $u(2^j t) \in V_{-1}$ by the previous paragraph. Reapplying Lemma 5.4, this time with $p = j$ and $q = j-1$, shows that $u(t) \in V_{j-1}$. □

Iterating Lemma 5.5 gives the inclusion ordering:

$$\cdots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \subset \cdots$$

Lemma 5.6 *If $\{\phi(t-k) : k \in \mathbf{Z}\}$ is a basis for V_0, then $\{\phi(2^j t - k) : k \in \mathbf{Z}\}$ is a basis for V_j. Furthermore, if the first is an orthogonal basis, then so is the second.*

Proof: Since $V_j = \text{span}\,\{\phi(2^{-j}t-k) : k \in \mathbf{Z}\}$, it is only necessary to check for linear independence. But if $0 = u(t) \stackrel{\text{def}}{=} \sum_k s(k)\phi(2^{-j}t-k) \in V_j$, then $v(t) \stackrel{\text{def}}{=} u(2^j t) \in V_0$ is also 0, and $0 = v(t) = \sum_k s(k)\phi(t-k)$ implies that $s(k) = 0$ for all $k \in \mathbf{Z}$, since, by hypothesis, $\{\phi(t-k) : k \in \mathbf{Z}\}$ is a linearly independent set.

For orthogonality, we have $\langle \phi(2^{-j}t-k), \phi(2^{-j}t-k') \rangle = 2^j \langle \phi(t-k), \phi(t-k') \rangle$ by the change of variable $t \leftarrow 2^j t$. One inner product is zero if and only if the other is zero, since 2^j never vanishes. Thus, if $\phi(t-k)$ is orthogonal to $\phi(t-k')$ whenever $k' \neq k$, the same holds for $\phi(2^{-j}t-k)$ and $\phi(2^{-j}t-k')$. □

Abstracting this construction and adding some properties defines a *multiresolution analysis* of $L^2(\mathbf{R})$, or *MRA*. This is a chain of subspaces $\{V_j : j \in \mathbf{Z}\}$ of $L^2(\mathbf{R})$ satisfying the following:

MRA Conditions

Containment: $V_j \subset V_{j-1} \subset L^2$ for all $j \in \mathbf{Z}$;

Decrease: $\lim_{j\to\infty} V_j = 0$; that is, $\bigcap_{j>N} V_j = \{0\}$ for all N;

Increase: $\lim_{j\to-\infty} V_j = L^2$, that is, $\bigcup_{j<N} V_j$ is dense in L^2 for all N;

Dilation: $v(2t) \in V_{j-1} \iff v(t) \in V_j$;

Scaling Function: There is a function $\phi \in V_0$, integrable and square integrable, having compact support and nonzero integral, whose integer translates $\{\phi(t-k) : k \in \mathbf{Z}\}$ form a Schauder basis for V_0.

Notice that $\{2^{-j/2}\phi(2^{-j}t-k) : k \in \mathbf{Z}\}$ will be a Schauder basis for V_j.

The scaling function is sometimes called a *father function*, to emphasize its relation to the mother function of a wavelet basis. If the scaling function generates an orthonormal basis for V_0, then we have an *orthogonal MRA*. In that case, $\{2^{-j/2}\phi(2^{-j}t-k) : k \in \mathbf{Z}\}$ will be an orthonormal basis for V_j. The factor $2^{-j/2}$ gives the basis functions unit norm.

Taking $V_j \to L^2$ as $j \to -\infty$ is the "Daubechies" indexing convention for an MRA. In it, the scale or width of the building blocks increases with j. It corresponds directly to the indices computed by the programs in this text.

The alternative or "Mallat" indexing convention reverses the sense of scale: $V_j \to L^2$ as $j \to +\infty$. In that convention, the resolution, or narrowness, of a building block in V_j increases with j. The two conventions are equivalent and produce equal sets of coefficients; they differ only in how the coefficients are tagged for subsequent processing.

5.2.2 From MRAs to filters

Now suppose that we have an orthogonal MRA with a scaling function ϕ, which we will insist has compact support, giving an orthonormal Schauder basis $\mathbf{B}_0 = \{\phi(t-k) : k \in \mathbf{Z}\}$ for $V_0 = \operatorname{span}\mathbf{B}_0$. The containment condition $V_0 \subset V_{-1}$ implies that ϕ may be expanded in the orthonormal Schauder basis $\mathbf{B}_{-1} = \{\sqrt{2}\phi(2t-k) : k \in \mathbf{Z}\}$ of $V_{-1} = \operatorname{span}\mathbf{B}_{-1}$. Writing the expansion coefficients $h(k) = \langle\sqrt{2}\phi(2t-k), \phi(t)\rangle$, we see that ϕ must satisfy a two-scale relation:

$$\phi(t) = \sum_k h(k)\sqrt{2}\,\phi(2t-k) = H\phi(t). \tag{5.36}$$

This defines a filter sequence h and transformation H. The compact support of ϕ implies that $h(k) = 0$ for all sufficiently large $|k|$, so h will be an FIR filter. Coming from an orthogonal MRA, this filter satisfies some special properties.

First, orthonormality of the set $\mathbf{B}_0 \subset L^2(\mathbf{R})$ implies that, for $n \neq 0$,

$$\begin{aligned} 0 = \langle\phi(t), \phi(t-n)\rangle &= \sum_l \sum_k \overline{h(l)}h(k)\left\langle\sqrt{2}\,\phi(2t-l), \sqrt{2}\,\phi(2t-2n-k)\right\rangle \\ &= \sum_l \sum_k \overline{h(l)}h(k)\delta(2n+k-l) = \sum_k \overline{h(2n+k)}h(k), \end{aligned}$$

and

$$\begin{aligned} 1 = \langle\phi(t), \phi(t)\rangle &= \sum_l \sum_k \overline{h(l)}h(k)\left\langle\sqrt{2}\,\phi(2t-l), \sqrt{2}\,\phi(2t-k)\right\rangle \\ &= \sum_l \sum_k \overline{h(l)}h(k)\delta(k-l) = \sum_k |h(k)|^2. \end{aligned}$$

Here δ is the Kronecker symbol of Equation 2.31:

$$\delta(n) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } n = 0; \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the filter h determined by an MRA satisfies the *self-orthonormality condition*

$$\sum_k |h(k)|^2 = 1; \qquad \sum_k \overline{h(k)} h(k+2n) = 0, \quad \text{if } n \neq 0. \tag{5.37}$$

If h is real valued, the complex conjugation is unnecessary.

Second, since ϕ is integrable, Equation 5.36 can be integrated as follows:

$$\int \phi(t)\,dt = \sqrt{2}\sum_k h(k) \int \phi(2t-k)\,dt = \frac{1}{\sqrt{2}}\sum_k h(k) \int \phi(t)\,dt, \tag{5.38}$$

where the substitutions $t \leftarrow (t+k)/2$ have been performed on the middle. Dividing by the nonzero $\int \phi(t)\,dt$ shows that

$$\sum_k h(k) = \sqrt{2}. \tag{5.39}$$

More is true: since $2 = |\sum_k h(k)|^2 = \sum_k \sum_n \overline{h(k)} h(n)$, we can substitute $n \leftarrow k+n$ and then divide into even n and odd n to have

$$2 = \sum_n \sum_k \overline{h(k)} h(k+2n) + \sum_n \sum_k \overline{h(k)} h(k+2n+1).$$

The first sum is 1 by Equation 5.37, leaving

$$\begin{aligned} 1 &= \sum_n \sum_k \overline{h(k)} h(k+2n+1) = \sum_k \overline{h(k)} \sum_n h(k+2n+1) \\ &= \left(\sum_k \overline{h(2k)}\right)\left(\sum_n h(2n+1)\right) + \left(\sum_k \overline{h(2k+1)}\right)\left(\sum_n h(2n)\right), \end{aligned}$$

after splitting the k-sum into even and odd k, and substituting $k \leftarrow 2k$, $n \leftarrow n-k$ in the even-k part, and $k \leftarrow 2k+1$, $n \leftarrow n-k-1$ in the odd-k part. Writing $E = \sum_k h(2k)$ and $O = \sum_k h(2k)$, this means $1 = \bar{E}O + E\bar{O}$. But since $E+O = \sqrt{2}$ by Equation 5.39,

$$|E-O|^2 = |E+O|^2 - 2(\bar{E}O + E\bar{O}) = 2 - 2 = 0,$$

so $E = O = 1/\sqrt{2}$. This is the *low-pass filter condition:*

$$\sum_k h(2k) = \sum_k h(2k+1) = \frac{1}{\sqrt{2}} \qquad \left(\Rightarrow \sum_k h(k) = \sqrt{2}\right). \tag{5.40}$$

Third, we observe that another filter with similar orthogonality properties can be defined from h:

$$g(k) = (-1)^k \overline{h(1-k)}, \qquad \text{for all } k \in \mathbf{Z}. \tag{5.41}$$

Clearly, g will be finite whenever h is finite, and given g we may determine h by the similar formula $h(k) = (-1)^{1-k} g(1-k)$. This and Equation 5.40 implies the *high-pass filter condition* for g:

$$\sum_k g(2k) = -\sum_k g(2k+1) = \frac{1}{\sqrt{2}} \qquad \left(\Rightarrow \sum_k g(k) = 0 \right). \tag{5.42}$$

Fourth, there is a *self-orthonormality condition* for g:

$$\begin{aligned} \sum_k \overline{g(k)} g(k+2n) &= \sum_k (-1)^k h(1-k)(-1)^{k+2n} \overline{h(1-k-2n)} \\ &= \sum_k h(1-k)\overline{h(1-k-2n)} \\ &= \sum_k h(k)\overline{h(k-2n)} \quad = \delta(n). \end{aligned} \tag{5.43}$$

Fifth, for any integer n, the following *independence condition* holds between the two filters h and g:

$$\begin{aligned} \sum_k \overline{g(k)} h(k+2n) &= \sum_k (-1)^k h(1-k) h(k+2n) \qquad (5.44) \\ &= \sum_{\text{even } k} h(1-k)h(k+2n) - \sum_{\text{odd } k} h(1-k)h(k+2n) \\ &= \sum_p h(2p+1)h(2n-2p) - \sum_q h(2n-2q)h(2q+1) \;=\; 0. \end{aligned}$$

Here $k \leftarrow -2p$ in the first sum and $k \leftarrow 2q+1-2n$ in the second.

Finally, the filter pair h, g satisfies the *completeness* condition:

$$\sum_k h(n-2k)h(m-2k) + g(n-2k)g(m-2k) = \delta(n-m). \tag{5.45}$$

This can be shown case-by-case. We first write g in terms of h, making the sum

$$\sum_k h(2k+n)h(2k+m) + (-1)^{n+m} \sum_k h(2k+1-n)h(2k+1-m).$$

Then we put $p = m-n$ to have $n+m = 2n+p$ and $(-1)^{n+m} = (-1)^p$, and consider the cases:

- If $n = 2n'$ is even, then substituting $k \leftarrow k-n'$ in the first sum and $k \leftarrow k+n'$ in the second reduces them to

$$\sum_k h(2k)h(2k+p) + (-1)^p \sum_k h(2k+1)h(2k+1-p).$$

- If $p = 2p'$ is even, then substituting $k \leftarrow k + p'$ in the second sum gives

$$\begin{aligned}\sum_k h(2k)h(2k+p) + \sum_k h(2k+1+p)h(2k+1) &= \sum_k h(k)h(k+p) \\ = \sum_k h(k)h(k+2p') &= \delta(p') = \delta(n-m).\end{aligned}$$

- If $p = 2p' + 1$ is odd, then substituting $k \leftarrow k + p'$ in the second sum gives

$$\sum_k h(2k)h(2k+p) - \sum_k h(2k+p)h(2k) = 0.$$

This agrees with the value of $\delta(n-m)$, which is 0 in this case since $p = m - n$ being odd means $n \neq m$.

- If $n = 2n' + 1$ is odd, then substituting $k \leftarrow k - n'$ in the first sum and $k \leftarrow k + n'$ in the second reduces them to

$$\sum_k h(2k+1)h(2k+1+p) + (-1)^p \sum_k h(2k)h(2k-p).$$

- If $p = 2p'$ is even, then substituting $k \leftarrow k + p'$ in the second sum gives

$$\begin{aligned}\sum_k h(2k+1)h(2k+1+p) + \sum_k h(2k+p)h(2k) &= \sum_k h(k)h(k+p) \\ = \sum_k h(k)h(k+2p') &= \delta(p') = \delta(n-m).\end{aligned}$$

- If $p = 2p' - 1$ is odd, then substituting $k \leftarrow k + p'$ in the second sum gives

$$\sum_k h(2k+1)h(2k+1+p) - \sum_k h(2k+p+1)h(2k+1) = 0.$$

This agrees with the value of $\delta(n-m)$, which is 0 in this case since $p = m - n$ being odd means $n \neq m$.

The sequences h and g derived from the MRA are called *orthogonal conjugate quadrature filters*, or orthogonal CQFs. We may abstract the properties just deduced from the MRA conditions:

Orthogonal CQF Conditions (Basic)

Finiteness: Sequence $h = \{h(k) : k \in \mathbf{Z}\}$ consists of zeroes for all but finitely many values of k.

Normalization of h: $\sum_k h(2k) = \sum_k h(2k+1) = 1/\sqrt{2}$, and thus $\sum_k h(k) = \sqrt{2}$.

Self-Orthonormality of h: $\sum_k h(k+2n)h(k+2m) = \delta(n-m)$, for every $n, m \in \mathbf{Z}$.

From these stand-alone assumptions, the other properties of h and g can be deduced:

Orthogonal CQF Conditions (Derived)

Conjugacy: There is a fixed integer M, and $g(k) = (-1)^k h(2M+1-k)$, for $k \in \mathbf{Z}$, defines another (finitely-supported) sequence $g = \{g(k) : k \in \mathbf{Z}\}$.

Normalization of g: $\sum_k g(2k) = -\sum_k g(2k+1) = 1/\sqrt{2}$, and thus $\sum_k g(k) = 0$.

Self-Orthonormality of g: $\sum_k g(k+2n)g(k+2m) = \delta(n-m)$.

Independence: $\sum_k g(k+2n)h(k+2m) = 0$ for any $n, m \in \mathbf{Z}$.

Completeness: $\sum_k h(2k+n)h(2k+m) + \sum_k g(2k+n)g(2k+m) = \delta(n-m)$.

The so-called *lazy filters*, $h(k) = \sqrt{2}\,\delta(k-1)$ and $g(k) = \sqrt{2}\,\delta(k)$ satisfy the finiteness, conjugacy, self-orthonormality, independence and completeness conditions, but only part of the normalization conditions. This partial example is a useful test case for some constructions.

To be definite, suppose that for some fixed $L > 0$, $h(k) = 0$ if $k < 0$ or $k \geq L$; this may be called *conventional indexing*. Then the length of the finite support of h is no more than L. If it is exactly L, namely if $h(0) \neq 0$ and $h(L-1) \neq 0$, then h is said to have *filter length* L. The normalization condition implies that filter length $L \geq 2$.

Lemma 5.7 *An orthogonal conjugate quadrature filter's length must be even.*

Proof: It is enough to prove this for the low-pass filter h, since the high-pass conjugate filter g will have the same length L as h. If $L = 2l+1$ for $l > 0$, then $L - 1 = 2l$ is the largest index k for which $h(k) \neq 0$, so

$$0 = \sum_k h(k)h(k+2l) = h(0)h(2l) = h(0)h(L-1).$$

Thus either $h(0) = 0$ or $h(L-1) = 0$, contradicting the assumption that h has length L. □

Constructing orthogonal filter pairs

How can we construct a finite sequence $h = \{h(k) : k \in \mathbf{Z}\}$ satisfying the orthogonal CQF conditions?

One solution can be found right away, the *Haar filter*, which is the unique orthogonal CQF of length two:

$$h(k) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } k = 0 \text{ or } k = 1, \\ 0, & \text{if } k \notin \{0,1\}; \end{cases} \qquad g(k) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } k = 0, \\ -\frac{1}{\sqrt{2}}, & \text{if } k = 1, \\ 0, & \text{if } k \notin \{0,1\}. \end{cases} \tag{5.46}$$

Filters of length four are not unique. Let h be an orthogonal CQF with nonzero coefficients $h(0)$, $h(1)$, $h(2)$, and $h(3)$. Then h must satisfy the norm condition $h^2(0) + h^2(1) + h^2(2) + h^2(3) = 1$, plus the following constraints:

$$h(0) + h(2) = \frac{1}{\sqrt{2}}; \qquad h(1) + h(3) = \frac{1}{\sqrt{2}}; \qquad h(0)h(2) + h(1)h(3) = 0. \quad (5.47)$$

By the first two conditions, picking $h(0)$ and $h(1)$ determines $h(2) = \frac{1}{\sqrt{2}} - h(0)$ and $h(3) = \frac{1}{\sqrt{2}} - h(1)$. The third condition holds if and only if there is some constant c for which $h(2) = ch(1)$ and $h(3) = -ch(0)$. The result is a system of two linear equations for $h(0)$ and $h(1)$, containing a free parameter c:

$$\begin{array}{rcl} h(0) + ch(1) & = & \frac{1}{\sqrt{2}} \\ -ch(0) + h(1) & = & \frac{1}{\sqrt{2}} \end{array} \quad \Rightarrow \quad \begin{pmatrix} 1 & c \\ -c & 1 \end{pmatrix} \begin{pmatrix} h(0) \\ h(1) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

The matrix is nonsingular for every real c since its determinant, $1 + c^2$, is at least one, and the one-parameter set of solutions is obtainable by inverting:

$$\begin{pmatrix} h(0) \\ h(1) \end{pmatrix} = \frac{1}{\sqrt{2}(1+c^2)} \begin{pmatrix} 1 & -c \\ c & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \Rightarrow \quad \begin{array}{rcl} h(0) & = & \frac{1-c}{\sqrt{2}(1+c^2)} \\ h(1) & = & \frac{1+c}{\sqrt{2}(1+c^2)} \end{array} \quad (5.48)$$

The remaining coefficients are then $h(2) = \frac{c(c+1)}{\sqrt{2}(1+c^2)}$ and $h(3) = \frac{c(c-1)}{\sqrt{2}(1+c^2)}$.

The *Daubechies 4 filters* are obtained this way using $c = 2 - \sqrt{3}$:

$$h(0) = \frac{1+\sqrt{3}}{4\sqrt{2}}; \quad h(1) = \frac{3+\sqrt{3}}{4\sqrt{2}}; \quad h(2) = \frac{3-\sqrt{3}}{4\sqrt{2}}; \quad h(3) = \frac{1-\sqrt{3}}{4\sqrt{2}}. \quad (5.49)$$

The normalization condition for 5.47 seems to impose an additional constraint on c. However, that condition is satisfied for all real c:

$$\begin{array}{rcl} h^2(0)+h^2(1)+h^2(2)+h^2(3) & = & (1+c^2)(h^2(0)+h^2(1)) \\ & = & (1+c^2)\left(\frac{1-2c+c^2}{2(1+c^2)^2} + \frac{1+2c+c^2}{2(1+c^2)^2}\right) \; = \; 1. \end{array}$$

If all four coefficients are nonzero, then $c \notin \{0, \pm 1, \pm\infty\}$. Otherwise, the degenerate cases are

$$\begin{array}{llll} c = -1 & \Rightarrow & h = \{\frac{1}{\sqrt{2}}, 0, 0, \frac{1}{\sqrt{2}}\}; & c = 0 \quad \Rightarrow \quad h = \{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0\}; \\ c = 1 & \Rightarrow & h = \{0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\}; & c = \pm\infty \quad \Rightarrow \quad h = \{0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\}. \end{array}$$

These are all just variations on the Haar filters.

Mother functions and details

The conjugate filter g derived from h defines the *mother function* for the MRA by way of a linear transformation G:

$$\psi(t) = \sum_k g(k)\sqrt{2}\,\phi(2t-k) \stackrel{\text{def}}{=} G\phi(t). \quad (5.50)$$

Thus $\psi(t-k) \in V_{-1}$ for every integer k. In fact, $\{\psi(t-k) : k \in \mathbf{Z}\}$ is an orthonormal subset of V_{-1}:

$$\begin{aligned}\langle \psi(t-n), \psi(t-m) \rangle &= \\ &= \sum_l \sum_k \overline{g(l)} g(k) \left\langle \sqrt{2}\,\phi(2t-2n-l), \sqrt{2}\,\phi(2t-2m-k) \right\rangle \\ &= \sum_l \sum_k \overline{g(l)} g(k) \delta(2n+l-2m-k) \\ &= \sum_k \overline{g(2(m-n)+k)} g(k) \qquad = \delta(n-m).\end{aligned}$$

This mother function defines another collection of subspaces in the MRA. Put $W_0 = \text{span}\,\{\psi(t-k) : k \in \mathbf{Z}\}$, and observe that $W_0 \subset V_{-1}$. In general, for any integer j, put

$$W_j \stackrel{\text{def}}{=} \text{span}\,\{\psi(2^{-j}t-k) : k \in \mathbf{Z}\}. \tag{5.51}$$

Then $W_j \subset V_{j-1}$. Note that $\{2^{-j/2}\psi(2^{-j}t-k) : k \in \mathbf{Z}\}$ is an orthonormal basis for W_j.

By the independence condition, every basis vector of W_0 is orthogonal to every basis vector of V_0:

$$\begin{aligned}\langle \psi(t-n), \phi(t-m) \rangle &= \\ &= \sum_l \sum_k \overline{g(l)} h(k) \left\langle \sqrt{2}\,\phi(2t-2n-l), \sqrt{2}\,\phi(2t-2m-k) \right\rangle \\ &= \sum_l \sum_k \overline{g(l)} h(k) \delta(2n+l-2m-k) \\ &= \sum_k \overline{g(2(n-m)+k)} h(k) \quad = 0.\end{aligned}$$

Also, since $\langle \psi(t-n), \phi(t-m) \rangle = 2^j \left\langle \psi(2^j t-n), \phi(2^j t-m) \right\rangle$ for every $j \in \mathbf{Z}$, every basis vector of W_j is orthogonal to every basis vector of V_j. In other words, $W_j \perp V_j$.

Multiresolution analysis works because $f \in V_{-1}$ is the sum of an average part that lies in V_0 and a complementary detail part that lies in W_0:

Lemma 5.8 $W_0 + V_0 = V_{-1}$.

Proof: We first show that each basis function of V_{-1} is a sum of a function in V_0 and a function in W_0, namely, that

$$\sqrt{2}\phi(2t-n) = \sum_k h(n-2k)\phi(t-k) + g(n-2k)\psi(t-k). \tag{5.52}$$

Using the two-scale relations for the scaling and mother functions, we may expand the ϕ and ψ terms. Then we use Equation 5.45, the completeness condition, to

evaluate the sum over index k as follows:

$$\begin{aligned}
\sum_k h(n-2k)\phi(t-k) + g(n-2k)\psi(t-k) \;&= \\
&= \sum_k \sum_m h(n-2k)h(m)\sqrt{2}\phi(2t-2k-m) \\
&\qquad + g(n-2k)g(m)\sqrt{2}\phi(2t-2k-m) \\
&= \sum_m \left(\sum_k h(n-2k)h(m-2k) + g(n-2k)g(m-2k) \right) \sqrt{2}\phi(2t-m) \\
&= \sum_m \delta(n-m)\sqrt{2}\phi(2t-m) \quad = \sqrt{2}\phi(2t-n).
\end{aligned}$$

Thus, for any $u = u(t) = \sum_k c(k)\sqrt{2}\phi(2t-k) \in V_{-1}$, there is a function $P_0u(t) \stackrel{\text{def}}{=} \sum_k s(k)\phi(t-k) \in V_0$, where $s(k) = \langle \phi(t-k), u(t) \rangle$, and a function $Q_0u(t) \stackrel{\text{def}}{=} \sum_k d(k)\psi(t-k) \in W_0$, where $d(k) = \langle \psi(t-k), u(t) \rangle$, and since $c(n) = \sum_k h(n-2k)s(k) + \sum_k g(n-2k)d(k)$, it follows that $u = P_0u + Q_0u$. □

This decomposition generalizes to arbitrary scales in the MRA. For fixed $j \in \mathbf{Z}$, define the functions

$$\phi_{jk}(t) \stackrel{\text{def}}{=} 2^{-j/2}\phi(2^{-j}t - k), \qquad k \in \mathbf{Z}, t \in \mathbf{R} \tag{5.53}$$

$$\psi_{jk}(t) \stackrel{\text{def}}{=} 2^{-j/2}\psi(2^{-j}t - k), \qquad k \in \mathbf{Z}, t \in \mathbf{R} \tag{5.54}$$

These are orthonormal basis vectors for V_j and W_j, respectively.

Corollary 5.9 *For every integer j, $W_j + V_j = V_{j-1}$.*

Proof: We substitute $t \leftarrow 2^{-j}t$ and multiply by $2^{-j/2}$ everywhere in Equation 5.52 in the proof of Lemma 5.8, then apply the definitions of ϕ_{jk} and ψ_{jk} to get

$$\phi_{j-1,n}(t) = \sum_k h(n-2k)\,\phi_{jk}(t) + g(n-2k)\,\psi_{jk}(t). \tag{5.55}$$

We have thus written an arbitrary basis function of V_{j-1} as a linear combination of basis functions of V_j and W_j. □

The subspaces W_j are the differences between the adjacent V_j and V_{j-1}. Knowing the expansion coefficients of u's approximation in V_j, it is only necessary to get the expansion coefficients of its projection on W_j (and to do some arithmetic) in order to get a better approximation of u in V_{j-1}. We may call W_j a *detail* space, since it contains the details from u's approximation in V_{j-1} which are missing in V_j. Repeated application of this splitting yields the *discrete wavelet decomposition*:

Corollary 5.10 $V_0 = W_1 + W_2 + \cdots + W_J + V_J$, *for any integer $J > 0$.* □

If the scale and detail spaces form an orthogonal MRA, then the subspaces in the sum are pairwise orthogonal.

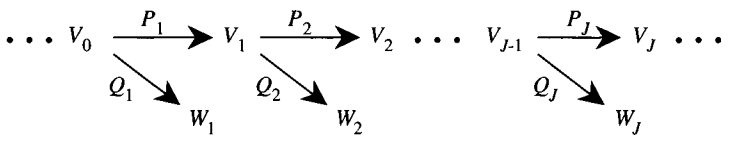


Figure 5.4: Pyramid of projections onto subspaces in the MRA.

5.2.3 From filters to discrete wavelet transforms

Once an orthogonal MRA has been fixed, a function u may be approximated to any desired degree in one of the scale spaces, $\{V_j : j \in \mathbf{Z}\}$, using the scaling function ϕ. We define a linear transformation P_j that produces the approximation by the following formula:

$$P_j u(t) = \sum_k \langle \phi_{jk}, u \rangle \, \phi_{jk}(t), \tag{5.56}$$

where $\phi_{jk}(t) = 2^{-j/2}\phi(2^{-j}t - k)$, as defined in Equation 5.53. If $u = u(t)$ has compact support, then the coefficient $s_j(k) \stackrel{\text{def}}{=} \langle \phi_{jk}, u \rangle$ is nonzero for only finitely many values of k, so $P_j u \in V_j$. Furthermore, $P_j(P_j u) = P_j u$, and $u - P_j u$ is orthogonal to $P_j u$, so P_j is the orthogonal projection onto V_j. The finitely-supported sequence of coefficients $\{s_0(k) : k \in \mathbf{Z}\}$, in particular, contains enough information to recover the approximation $P_0 u \in V_0$ of u. For integer $J > 0$, the sequence $\{s_J(k) : k \in \mathbf{Z}\}$ contains the smaller amount of information needed to recover the cruder approximation $P_J u \in V_J$.

Likewise, the orthogonal projection Q_j onto W_j is defined by

$$Q_j u(t) = \sum_k \langle \psi_{jk}, u \rangle \, \psi_{jk}(t), \tag{5.57}$$

where ψ is the mother function of the orthogonal MRA, and $\psi_{jk}(t)$ is defined in Equation 5.54. Again, coefficient $d_j(k) \stackrel{\text{def}}{=} \langle \psi_{jk}, u \rangle$ will be nonzero for only finitely many values of k, whenever u has compact support. The finitely-supported sequences $\{d_1(k) : k \in \mathbf{Z}\}$, $\{d_2(k) : k \in \mathbf{Z}\}$,..., $\{d_J(k) : k \in \mathbf{Z}\}$, contain the information from which we may recover the details in spaces W_1, W_2,..., W_J. This decomposition,

$$\begin{aligned} V_0 &= W_1 + W_2 + \cdots + W_J + V_J; \\ &\updownarrow \\ P_0 u &= Q_1 u + Q_2 u + \cdots + Q_J u + P_J u, \end{aligned}$$

is depicted in Figure 5.4.

The orthogonal CQF properties imply that

$$s_{j+1}(n) = \sum_k h(k) s_j(2n + k) = \sum_k h(k - 2n) s_j(k); \tag{5.58}$$

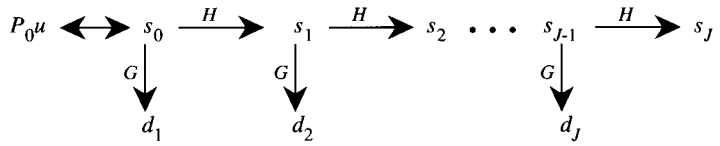


Figure 5.5: The pyramid algorithm for the discrete wavelet transform (DWT).

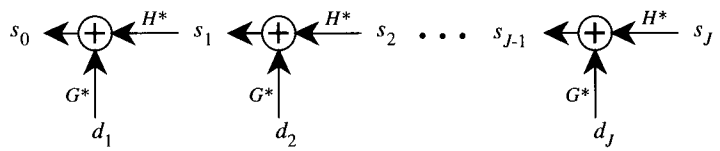


Figure 5.6: The pyramid algorithm for the inverse discrete wavelet transform (iDWT).

$$d_{j+1}(n) = \sum_k g(k)s_j(2n+k) = \sum_k g(k-2n)s_j(k); \tag{5.59}$$

$$s_{j-1}(n) = \sum_k h(n-2k)s_j(k) + g(n-2k)d_j(k). \tag{5.60}$$

Equations 5.58 and 5.59 provide a recursive algorithm, the *Mallat pyramid algorithm* depicted in Figure 5.5, for computing expansions in any of the subspaces of the MRA. The finitely-supported sequence $s_0 = \{s_0(k) : k \in \mathbf{Z}\}$, defined by $s_0(k) = \langle \phi(t-k), u(t) \rangle$, completely determines the approximation $P_0u \in V_0$. The *wavelet expansion of* P_0u *to level* $J > 0$ consists of the sequences $(d_1, d_2, \ldots, d_J; s_J)$. Sequence s_J determines the crude approximation $P_Ju \in V_J$, while d_J, d_{J-1}, and so on contain extra information needed to refine it successively into the better approximation $P_0u \in V_0$.

Reconstruction from the wavelet expansion is done by a similar pyramid algorithm, depicted in Figure 5.6. The arrows are reversed using adjoints, and the results are summed according to Equation 5.60.

In both pyramid algorithms, sequences $s_1, s_2, \ldots, s_{J-1}$ are computed along the way, even though they are not part of the discrete wavelet expansion or the reconstructed signal. Enough temporary storage to hold another copy of the signal may therefore be needed, depending upon how the arrow operations are implemented.

Filter transforms

A finitely-supported sequence $f = \{f(k) : k \in \mathbf{Z}\}$ defines a *filter transform*, acting on arbitrary sequences $u = \{u(k) : k \in \mathbf{Z}\}$ by either of the two equivalent formulas,

related by the substitution $k \leftarrow k' + 2n$:

$$Fu(n) = \sum_k f(k-2n)u(k) = \sum_{k'} f(k')u(k'+2n), \qquad n \in \mathbf{Z}. \tag{5.61}$$

This F is a linear transformation on every vector space of complex-valued sequences, including the inner product space ℓ^2 of square-summable sequences with inner product $\langle u, v\rangle = \sum_k \overline{u(k)}v(k)$. In that space, F has an adjoint F^* that satisfies $\langle Fu, v\rangle = \langle u, F^*v\rangle$ for all u and v. But then,

$$\begin{aligned}\sum_n \overline{Fu(n)}v(n) &= \sum_n \left(\sum_k \bar{f}(k-2n)\bar{u}(k)\right) v(n) \\ &= \sum_k \left(\bar{u}(k)\sum_n \bar{f}(k-2n)v(n)\right) \quad \stackrel{\text{def}}{=} \quad \sum_k \bar{u}(k)F^*v(k),\end{aligned}$$

which defines two equivalent formulas for the adjoint filter transform:

$$\begin{aligned}F^*v(k) &= \sum_n \bar{f}(k-2n)v(n) \qquad (5.62)\\ &= \begin{cases} \sum_{n'} \bar{f}(2n')v(\frac{k}{2}-n'), & \text{if } k \in \mathbf{Z} \text{ is even,} \\ \sum_{n''} \bar{f}(2n''+1)v(\frac{k-1}{2}-n''), & \text{if } k \in \mathbf{Z} \text{ is odd.}\end{cases}\end{aligned}$$

These are related by the substitutions $n \leftarrow \frac{k}{2} - n'$ if k is even, and $n \leftarrow \frac{k-1}{2} - n''$ if k is odd.

Composing F and its adjoint gives

$$F^*Fu(j) = \sum_n \bar{f}(2n-j)Fu(n) = \sum_{n,k} \bar{f}(2n-j)f(2n-k)u(k). \tag{5.63}$$

Similarly,

$$FF^*u(m) = \sum_k f(2m-k)F^*u(k) = \sum_{k,n} f(2m-k)\bar{f}(2n-k)u(n). \tag{5.64}$$

Because of the dilation by 2, F typically shrinks the support of sequences, while F^* enlarges it:

Lemma 5.11 *Suppose that the sequence f is supported on the index interval $[a, b]$: $f = \{\ldots, 0, f(a), f(a+1), \ldots, f(b-1), f(b), 0, \ldots\}$, since $f(n) = 0$ if $n < a$ or $n > b$. For any sequence u supported on $[x, y]$,*

- *Fu is supported on $[\lceil \frac{x-b}{2} \rceil, \lfloor \frac{y-a}{2} \rfloor]$;*
- *F^*u is supported on $[2x + a, 2y + b]$.*

Proof: Taking into account the support, the second version of the filter transform formula reduces to

$$Fu(n) = \sum_{k'=a}^{b} f(k')u(k' + 2n). \tag{5.65}$$

Notice that the summand will be zero if $b + 2n < x$ or $a + 2n > y$. Only output values at indices $n \in [x', y']$ need to be computed, where $x' = \lceil (x-b)/2 \rceil$ and $y' = \lfloor (y-a)/2 \rfloor$.

On the other hand, the first version of the adjoint filter transform formula reduces to

$$F^* v(k) = \sum_{n=x}^{y} \bar{f}(k - 2n)v(n). \tag{5.66}$$

The summand will be zero unless $a \leq k - 2n \leq b$ and $x \leq n \leq y$. Hence, output values need only be computed at indices $k \in [x'', y'']$, where $x'' = 2x + a$ and $y'' = 2y + b$. □

Lemma 5.11 illuminates two kinds of *spreading in support* that occur with filter transforms. Firstly, if F is one filter from an orthogonal CQF pair, then F^*F is an orthogonal projection on ℓ^2, but the support of F^*Fu may be greater than that of u. If u is finitely-supported on the index interval $[x, y]$, and f is supported on $[a, b]$, then F^*Fu will be finitely supported in the index interval $\left[2\left\lceil \frac{x-b}{2} \right\rceil + a, 2\left\lfloor \frac{y-a}{2} \right\rfloor + b\right]$. This contains $[x - (b - a - 1), y + (b - a - 1)]$, which in turn contains $[x, y]$ and is strictly larger if and only if $b - a > 1$. The only orthogonal CQF with $b - a \leq 1$ is the Haar pair, with $a = 0, b = 1$ in the conventional indexing giving $b - a = 1$.

Secondly, a CQF pair H, G of filter transforms can produce more total output values than there are input values. Suppose the supports are $[a, b]$ for H and $[c, d]$ for G. For a finitely supported sequence $u = \{u(x), \ldots, u(y)\}$ of length $N = 1 + y - x$, the high-pass and low-pass parts of the signal will be supported on the intervals $\left[\left\lceil \frac{x-b}{2} \right\rceil, \left\lfloor \frac{y-a}{2} \right\rfloor\right]$ and $\left[\left\lceil \frac{x-d}{2} \right\rceil, \left\lfloor \frac{y-c}{2} \right\rfloor\right]$, respectively, with total length

$$\left(1 + \left\lfloor \frac{y-a}{2} \right\rfloor - \left\lceil \frac{x-b}{2} \right\rceil\right) + \left(1 + \left\lfloor \frac{y-c}{2} \right\rfloor - \left\lceil \frac{x-d}{2} \right\rceil\right). \tag{5.67}$$

The total support length will be greater than $1 + y - x$ if and only if $b - a > 1$ or $d - c > 1$. The Haar CQF pair has $b - a = 1$ and $d - c = 1$, and is the only filter pair that does not cause spreading of the support.

Successive applications of H and G give the (nonperiodic) discrete orthogonal wavelet transform on finitely-supported infinite sequences. There are only finitely many finitely-supported sequences $d_1, d_2, \ldots, d_J$, and s_J to compute, and each output coefficient costs only a finite number of operations since h, g are both finite sequences, say of length L. Since L must be even by Lemma 5.7, we can write $L = 2M$ for an integer M. If h is conventionally indexed, so that $h(k)$ is nonzero only for $0 \leq k < L$, then we may choose[3] to define $g(k) = (-1)^k h(2M - k - 1)$ to insure that $g(k)$ is also nonzero only for $0 \leq k < L$.

[3]Work Exercise 12 to see why!

With these indexing conventions, if $s_j(k)$ is supported in $x \leq k \leq y$, then $d_{j+1}(n)$ and $s_{j+1}(n)$ may be nonzero for $\lceil (1+x-L)/2 \rceil \leq n \leq \lfloor y/2 \rfloor$. Hence, the output sequences are of varying lengths:

Mallat's Discrete Wavelet Transform

```
dwt( u[], x, y, J, h[], g[], L ):
[0]  If J=0 then for n = x to y, print u[n]
[1]  Else do [2] to [9]
[2]     Let x1 = ceiling((1+x-L)/2), let y1 = floor(y/2)
[3]     For n=x1 to y1, do [4] to [8]
[4]        Let s[n] = 0, let d[n] = 0
[5]        For k=0 to L-1, do [6] to [7]
[6]           Accumulate s[n] += h[k]*u[k+2*n]
[7]           Accumulate d[n] += g[k]*u[k+2*n]
[8]        Print d[n]
[9]     Compute dwt( s[], x1, y1, J-1, h[], g[], L )
```

Of course, values `d[n]` do not have to be printed as soon as they are computed, they may be stored in an array. For fixed L and J, this array will require $O(N)$ total elements, and will cost $O(N)$ operations to fill.

Periodic filter transforms

If f_{2q} is a $2q$-periodic sequence with even period, then it can be used to define a *periodic filter transform* F_{2q} from $2q$-periodic to q-periodic sequences, and a *periodic adjoint* F_{2q}^* from q-periodic to $2q$-periodic sequences. These are, respectively, the transformations

$$F_{2q}u(n) = \sum_{k=0}^{2q-1} f_{2q}(k-2n)u(k) = \sum_{k'=0}^{2q-1} f_{2q}(k')u(k'+2n), \qquad 0 \leq i < q; \quad (5.68)$$

and

$$F_{2q}^* v(k) = \sum_{n=0}^{q-1} \bar{f}_{2q}(k-2n)v(n) \quad (5.69)$$

$$= \begin{cases} \sum_{n'=0}^{q-1} \bar{f}_{2q}(2n')v(\frac{k}{2} - n'), & \text{if } k \in [0, 2q-2] \text{ is even,} \\ \sum_{n''=0}^{q-1} \bar{f}_{2q}(2n''+1)v(\frac{k-1}{2} - n''), & \text{if } k \in [1, 2q-1] \text{ is odd.} \end{cases} \quad (5.70)$$

We have performed the same substitutions as in Equations 5.61 and 5.62. Except for the index ranges, the formulas are the same.

Periodization commutes with filter transforms: we get the same periodic sequence whether we first filter an infinite sequence and then periodizes the result,

or first periodize both the sequence and the filter and then perform a periodic filter transform. To be precise:

Lemma 5.12 $(Fu)_q = F_{2q}u_{2q}$ *and* $(F^*v)_{2q} = F^*_{2q}v_q$.

Proof: Note that

$$\begin{aligned}
(Fu)_q(n) &= \sum_{j=-\infty}^{\infty} Fu(n+qj) = \sum_{j=-\infty}^{\infty}\sum_{k=-\infty}^{\infty} f(k-2[n+qj])u(k) \\
&= \sum_{k=-\infty}^{\infty}\left(\sum_{j=-\infty}^{\infty} f(k-2n-2qj)\right)u(k) = \sum_{k=-\infty}^{\infty} f_{2q}(k-2n)u(k) \\
&= \sum_{k_1=0}^{2q-1}\sum_{k_2=-\infty}^{\infty} f_{2q}(k_1+2qk_2-2n)u(k_1+2qk_2) \\
&= \sum_{k_1=0}^{2q-1} f_{2q}(k_1-2n)\sum_{k_2=-\infty}^{\infty} u(k_1+2qk_2) \\
&= \sum_{k_1=0}^{2q-1} f_{2q}(k_1-2n)u_{2q}(k_1).
\end{aligned}$$

Also,

$$\begin{aligned}
(F^*v)_{2q}(k) &= \sum_{j=-\infty}^{\infty} F^*v(k+2qj) = \sum_{j=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} \bar{f}([k+2qj]-2n)v(n) \\
&= \sum_{n=-\infty}^{\infty}\left(\sum_{j=-\infty}^{\infty} \bar{f}(k+2qj-2n)\right)v(n) = \sum_{n=-\infty}^{\infty} \bar{f}_{2q}(k-2n)v(n) \\
&= \sum_{n_1=0}^{q-1}\sum_{n_2=-\infty}^{\infty} \bar{f}_{2q}(k-2n_1-2qn_2)v(n_1+qn_2) \\
&= \sum_{n_1=0}^{q-1} \bar{f}_{2q}(k-2n_1)\sum_{n_2=-\infty}^{\infty} v(n_1+qn_2) \\
&= \sum_{n_1=0}^{q-1} \bar{f}_{2q}(k-2n_1)v_q(n_1).
\end{aligned}$$

□

Thus, any redundancy introduced by periodizing an even-length input signal can be removed by ignoring all but one period of the output.

A $2q$-periodized pair of orthogonal CQFs h, g retain their orthogonal CQF properties. A $2q$-periodic input sequence u may be completely described by $2q$ coefficients, and H_{2q} and G_{2q} each produce q-periodic outputs that are completely

described by q coefficients each. It is efficient to combine the two periodic filter transforms as follows:

Combined Periodic Filter Transform on $2q$ Samples

```
pcqfilter( out[], in[], q, h[], g[], L ):
[0] For n=0 to q-1, do [1] to [4]
[1]    Let out[n] = 0, let out[n+q] = 0
[2]    For k=0 to L-1, do [3] to [4]
[3]       Accumulate  out[n]  += h[k]*in[(k+2*n)%(2*q)]
[4]       Accumulate out[n+q] += g[k]*in[(k+2*n)%(2*q)]
```

Both the input array `in[]` and the output array `out[]` must have $2q$ elements. The H output sequence $s(0), \ldots, s(q-1)$ gets written into the first q elements of `out[]`, namely `out[0]`,...,`out[q-1]`, and the G output sequence $d(0), \ldots, d(q-1)$ gets written into `out[q]`,...,`out[2*q-1]`. The index expression `(k+2*n)%(2*q)` gives a remainder in the range $0, \ldots, 2q-1$ for all values of $k + 2n$, as all its arguments are nonnegative.

Since applying a pair of CQFs together yields enough information to recover the original samples, the combined filter transform is invertible. With the arrangement of the s, d sequences produced by `pcqfilter()`, we may recover the original samples as follows:

Inverse of the Combined Periodic Filter Transform on $2q$ Samples

```
ipcqfilter( out[], in[], q, h[], g[], L ):
[0] For k2=0 to q-1, do [1] to [4+]
[1]   Let out[2*k2] = 0, let out[2*k2+1] = 0
[2]   For n2=0 to L/2-1, do [3] to [4+]
[3]     Sum  out[2*k2]  +=  h[2*n2]*in[(k2-n2) mod q]
[3+]    Sum  out[2*k2]  +=  g[2*n2]*in[((k2-n2) mod q) + q]
[4]     Sum out[2*k2+1] += h[2*n2+1]*in[(k2-n2) mod q]
[4+]    Sum out[2*k2+1] += g[2*n2+1]*in[((k2-n2) mod q) + q]
```

Since the arguments can be negative, the index expressions `(k2-n2) mod q` and `(1+k2-n2) mod q` must be implemented carefully to insure that they take values in the range $[0, q-1]$. The Standard C remainder operator can be used, if qL is added to each left-hand side to guarantee nonnegativity:

```
(k2-n2) mod q   ←   (q*L+k2-n2)%q
(1+k2-n2) mod q ←   (q*L+1+k2-n2)%q
```

Combined periodic filter transforms are *size preserving*, in that they produce as many output coefficients as input coefficients. Applying them recursively yields another kind of discrete wavelet transform. For $N = 2^J K$ with integer K and $J > 0$, output sequences s_j and d_j will both be periodic of period $N/2^j = 2^{J-j}K$, which is an integer if $j \leq J$. The total number of output wavelet expansion coefficients in the sequences $(d_1, d_2, \ldots, d_J; s_J)$ will equal the number of input coefficients, since

$\frac{N}{2^1} + \frac{N}{2^2} + \cdots + \frac{N}{2^J} + \frac{N}{2^J} = N.$

Mallat's Periodic Wavelet Transform

```
pdwt( u[], N, J, h[], g[], L ):
[0] If J>0, then do [1] to [4]
[1]    Allocate temp[0]=0,...,temp[N-1]=0
[2]    Compute pcqfilter( temp[], u[], N/2, h[], g[], L)
[3]    For i=0 to N-1, copy u[i] = temp[i]
[4]    Compute pdwt( u[], N/2, J-1, h[], g[], L )
```

Each output value $d_j(n)$ costs $LN/2^{j-1}$ multiply-adds to compute, plus there are another $LN/2^{j-1}$ multiply-adds needed to compute $s_j(n)$. The total cost is $2LN$, or $O(N)$ as $N \to \infty$, and is gotten by summing over $j = 1, 2, \ldots, J$.

An important special case is the periodic discrete orthogonal wavelet transform on $N = 2^J$ points, with complete expansion to the maximum depth J. The number of output coefficients equals the number of input coefficients, so the output can overwrite the input:

Complete Periodic Wavelet Transform on 2^J Samples

```
pdwt0( u[], J, h[], g[], L ):
[0] If J>0, then do [1-] to [4]
[1-]   Let N = 1<<J
[1]    Allocate temp[0]=0,...,temp[N-1]=0
[2]    Compute pcqfilter( temp[], u[], N/2, h[], g[], L )
[3]    For i=0 to N-1, copy u[i] = temp[i]
[4]    Compute pdwt0( u[], J-1, h[], g[], L )
```

Sequences s_J and d_J are each 1-periodic and representable by a single coefficient, which is in `u[0]` and `u[1]` of the output, respectively. The remaining sequences $d_{J-1}, \ldots, d_2, d_1$ are of lengths $2^1, \ldots, 2^{J-2}, 2^{J-1}$, respectively, and are found at consecutive indices starting with `u[2]`. It is straightforward to check that the original 2^J locations in `u[]` exactly suffice to hold everything. Along the way, an additional 2^J temporary memory locations are used.

Given wavelet expansion coefficients produced by `pdwt0()`, we can reconstruct the original signal samples as follows:

Reconstruction from Periodic Wavelet Expansion on 2^J Samples

```
ipdwt0( u[], J, h[], g[], L ):
[0] If J>0, then  do [1-] to [4]
[1-]   Let N = 1<<J
[1]    Compute ipdwt0( u[], J-1, h[], g[], L )
[2]    Allocate temp[0]=0,...,temp[N-1]=0
[3]    Compute ipcqfilter( temp[], u[], N/2, h[], g[], L )
[4]    For i=0 to N-1, let u[i] = temp[i]
```

Output is written over the input array `u[]`, as the inverse transform is size preserving.

We now use induction on J to prove that `pdwt0(u[],J,h[],g[],L)` is inverted by `ipdwt0(u[],J,h[],g[],L)`, for any orthogonal CQFs h, g.

For $J = 0$, both `pdwt0(u[],0,h[],g[],L)` and `ipdwt0(u[],0,h[],g[],L)` are the identity on one-point signals u, and are thus trivially inverses.

For $J > 0$, suppose that `ipdwt0(.,J-1,...)` inverts `pdwt0(.,J-1,...)` on all signals of length 2^{J-1}. Given a signal $\{u(k) : 0 \leq k < 2^J\}$ of length 2^J, consider `pdwt0(u[],J,h[],g[],L)`. This consists of `pcqfilter()` applied to u, followed by `pdwt0(u[],J-1,h[],g[],L))` which acts just on the first-half elements $\{u(k) : 0 \leq k < 2^{J-1}\}$. By the inductive hypothesis, that is inverted by `ipdwt0(u[],J-1,h[],g[],L)`, which recovers the first-half elements, followed by `ipcqfilter()` which combines those with $\{u(k) : 2^{J-1} \leq k < 2^J\}$ to recover the rest of u.

After implementation, it is a good idea to perform some tests with random sequences to see whether `ipdwt0()` and `pdwt0()` are inverses. The reconstructed samples will contain roundoff errors from the finite-precision arithmetic, but with orthogonal length-L CQFs the relative error should be about $2\epsilon_f J\sqrt{L}$.

Building multiresolutions from filters

Does choosing any h satisfying the stand-alone, or basic, orthogonal filter conditions on page 151 determine a multiresolution analysis? The scaling function ϕ is found by solving the two-scale equation. In turn, ϕ determines the entire MRA $\{V_j : j \in \mathbf{Z}\}$. Thus, we must show that the two-scale equation, for a given orthogonal CQF, always has a unique solution ϕ with all the scaling function properties.

One way to solve the two-scale equation is by iteration. Define a linear transformation from the sequence h:

$$Hf(t) = \sum_k h(k)\sqrt{2}f(2t-k). \tag{5.71}$$

The idea is to start with $f_0 = \mathbf{1}$ and apply H repeatedly: $f_{n+1} \stackrel{\text{def}}{=} Hf_n$, so

$$f_n \stackrel{\text{def}}{=} H^n\mathbf{1}, \quad \text{for } n = 0, 1, 2, \ldots. \tag{5.72}$$

Theorem 5.13 *If, for every t, $f_n(t)$ converges uniformly to a limit function $\phi(t)$ as $n \to \infty$, then*

1. *ϕ has compact support;*
2. *ϕ satisfies the two-scale equation: $H\phi = \phi$;*
3. *ϕ is nonzero, integrable, and satisfies $\int \phi(t)\,dt = 1$;*
4. *$\{\phi(t-k) : k \in \mathbf{Z}\}$ is an orthonormal set.*

Proof: Properties 1–4 will be shown in turn.

1. We show by induction that ϕ inherits compact support from **1**. Suppose $h(k) = 0$ for $k > b$ or $k < a$, and $f(t) = 0$ for $t \notin [c, d]$. Then $Hf(t) = \sqrt{2}\sum_{k=a}^{b} h(k)f(2t-k) = 0$ unless $c \le 2t - k \le d$ for some $k = a, a+1, \ldots, b$. But this means that $Hf(t) = 0$ unless $(a+c)/2 \le t \le (b+d)/2$. Iterating, $H^2 f(t) = 0$ unless $[a + (a+c)/2]/2 \le t \le [b + (b+d)/2]/2$, and $H^n f(t) = 0$ unless

$$(1-2^{-n})a + 2^{-n}c = \frac{a}{2} + \cdots + \frac{a}{2^n} + \frac{c}{2^n} \le t \le \frac{b}{2} + \cdots + \frac{b}{2^n} + \frac{d}{2^n} = (1-2^{-n})b + 2^{-n}d.$$

Note that if $t < -|a| - |c|$ or $t > |b| + |d|$, then $H^n f(t) = 0$ for every $n = 0, 1, 2, \ldots$. This means that u and all iterations $H^n f$ are supported in a single compact interval $[-|a| - |c|, |b| + |d|]$, of length $|a| + |b| + |c| + |d|$. Starting with **1**, with $c = 0$ and $d = 1$, the limit ϕ obtained as $n \to \infty$ is supported in $[a, b]$.

2. Let $\epsilon > 0$ be given. We show that $|\phi(t) - H\phi(t)| < \epsilon$ for every t, so since ϵ is arbitrary we must have $\phi = H\phi$. But for any $\delta > 0$, by uniform convergence we can choose N large enough so that $n \ge N \Rightarrow |f_n(t) - \phi(t)| < \delta$. Then, using the relation $f_{n+1}(t) = Hf_n(t)$ and the triangle inequality, we estimate

$$|\phi(t) - H\phi(t)| \le |\phi(t) - f_{n+1}(t)| + |Hf_n(t) - H\phi(t)| < \delta + B\delta,$$

where $B = \sqrt{2}\sum_k |h(k)|$ is the largest amount by which H can increase the maximum absolute value of a function. Now we go back and pick $\delta = \epsilon/(1+B)$.

3. First note that if u is integrable then so is Hf, since finite sums of integrable functions are integrable. Then observe that $\int Hf(t)\,dt = \int f(t)\,dt$ by Equation 5.38. Thus, for $n = 0, 1, 2, \ldots$, f_n is integrable and $\int f_n(t)\,dt = 1$. Now $f_n(t) \to \phi(t)$ uniformly at all t, so given $\epsilon > 0$ there is an N large enough so that $n \ge N \Rightarrow |\phi(t) - f_n(t)| < \epsilon$ for all t. Since all functions are supported in a single interval of length $L = |a| + |b| + |c| + |d|$, it follows[4] that ϕ is integrable, and

$$\left|\int \phi(t)\,dt - 1\right| = \left|\int \phi(t)\,dt - \int f_n(t)\,dt\right| \le \int |\phi(t) - f_n(t)|\,dt < \epsilon L.$$

Since $\left|\int \phi(t)\,dt - 1\right| < \epsilon L$ for every $\epsilon > 0$, it follows that $\int \phi(t)\,dt = 1$. Finally, ϕ must be nonzero because its integral is nonzero.

4. Suppose $\langle f(t-p), f(t-q)\rangle = \delta(p-q)$ for a compactly-supported function u. Then Hf inherits the same property:

$$\begin{aligned}\langle Hf(t-p), Hf(t-q)\rangle &= 2\int \sum_{k,k'} h(k-2p)h(k'-2q)f(2t-k)f(2t-k')\,dt \\ &= \sum_{k,k'} h(k-2p)h((k'-2q)\delta(k-k') \;=\; \delta(p-q).\end{aligned}$$

Thus, since **1** is orthogonal to its integer translates, so is f_n for every $n = 0, 1, \ldots$.

[4]Here we use the Lebesgue dominated convergence theorem, which is beyond the scope of this text. For a full proof, see Apostol, page 270, in the further readings from Chapter 3.

For n so large that $|\phi(t) - f_n(t)| < \epsilon$ at all t, the value $M = \int |\phi(t)|\, dt$ gives an upper bound $\int |f_n(t)|\, dt < M + \epsilon L$. Thus

$$\begin{aligned}
&\left| \int \phi(t-p)\phi(t-q)\, dt \;-\; \delta(p-q) \right| \;= \\
&\quad = \left| \int \phi(t-p)\phi(t-q)\, dt - \int f_n(t-p) f_n(t-q)\, dt \right| \\
&\quad \leq \int |\phi(t-p) - f_n(t-p)|\; |\phi(t-q)|\; dt \\
&\qquad\qquad + \int |f_n(t-p)|\; |\phi(t-q) - f_n(t-q)|\; dt \\
&\quad < \epsilon M + (M + \epsilon L)\epsilon.
\end{aligned}$$

Since this holds for all $\epsilon > 0$, it follows that $|\langle \phi(t-p), \phi(t-q) \rangle - \delta(p-q)| = 0$. This implies that $\langle \phi(t-p), \phi(t-q) \rangle = \delta(p-q)$, so the inner product is zero if $p \neq q$. Finally, setting $p = q = 0$ shows that $\|\phi\| = 1$. □

For the Haar filters of Equation 5.46, we have $H\mathbf{1} = \mathbf{1}$ as in Equation 5.31. Thus $f_N = \mathbf{1}$ for all $N = 0, 1, 2, \ldots$, so convergence is not only uniform but immediate: $\phi = \mathbf{1}$. However, a study of more general filters H, G that have uniform convergence of $\{f_n\}$ is beyond the scope of this book.[5]

To get lots of samples for a graph of a wavelet or scaling function, however, is easy. Since the wavelet transform of ϕ_{jk} has the expansion sequence $s_j = \mathbf{e}_k$, we simply apply the j-level inverse wavelet transform to the sequences $s_j = \mathbf{e}_k$; $d_j = d_{j-1} = \cdots = d_1 = \mathbf{0}$. Likewise, to get the samples for a graph of ψ_{jk}, we simply apply the j-level inverse wavelet transform to the sequences $s_j = \mathbf{0}$; $d_j = \mathbf{e}_k$; $d_{j-1} = \cdots = d_1 = \mathbf{0}$. We need to use the indexing formula for the particular wavelet transform to find the locations of $s_j(k)$ and $d_j(k)$.

5.2.4 Lifting

A clever method of implementing filter transforms is called *lifting*. The two output sequences (Hu, Gu) produced by a pair H, G of CQFs are computed together, efficiently and in a manner that reduces the amount of auxiliary storage. We illustrate with the example of Haar filters applied to a finitely-supported signal $u(0), \ldots, u(N-1)$ of length $N = 2q > 0$:

Lifting Implementation of the Haar Filter Transform on $2q$ Samples

```
haarlift( u[], q, dq ):
[1] For n=0 to q-1, replace u[(2*n+1)*dq] -=  u[(2*n)*dq]
[2] For n=0 to q-1, replace u[(2*n)*dq] += 0.5*u[(2*n+1)*dq]
[3] For n=0 to q-1, replace u[(2*n+1)*dq] /= sqrt(2.0)
[4] For n=0 to q-1, replace u[(2*n)*dq] *= sqrt(2.0)
```

[5] Cavaretta, Dahmen and Micchelli's *Stationary Subdivision*, in the further readings, has a highly detailed exposition of the relevant technicalities.

Ignore the increment dq for the moment, pretending that it is 1. Step 1 replaces all the odd-indexed elements of u with their differences $u(2n+1) - u(2n)$. Step 2 then adds half these differences into the even-indexed elements, leaving them containing the averages $[u(2n+1) + u(2n)]/2$. Steps 3 and 4 then normalize these differences and averages into the familiar Haar filter outputs $Gu(n) = [u(2n+1) - u(2n)]/\sqrt{2}$ and $Hu(n) = [u(2n+1) + u(2n)]/\sqrt{2}$.

Permitting increments $dq > 1$ makes it easier to use `haarlift()` in a recursive discrete wavelet transform. Note that $Hu(n)$ is found at index $(2n)dq$ of the output, and $Gu(n)$ is at index $(2n+1)dq$. This is different from the `pcqfilter()` convention, in which $Hu(n)$ is at index n, while $Gu(n)$ is at index $n+q$. Thus, a different recursion is needed to obtain the DWT by lifting:

Complete Discrete Haar Transform by Lifting on 2^J Samples

```
ldht0( u[], J, dq ):
[0] If J>0, then do [1] to [2]
[1]    Compute haarlift( u[], (1<<J)/2, dq )
[2]    Compute ldht0( u[], J-1, 2*dq )
```

Notice that no temporary storage array is required. Because of the way the outputs are interleaved, the locations of wavelet expansion coefficients $d_1, \ldots, d_J; s_J$ have more complicated index formulas than those of `pdwt0()`. In particular, $s_J(0)$ is found at `u[0]`, $d_J(0)$ is found at `u[(1<<J)/2]`. For $1 \le j \le J-1$, there are $N/2^j = 2^{J-j}$ coefficients in the sequence d_j, spaced $2dq = 2^j$ indices apart at the odd multiples of $dq = 2^{j-1}$. Thus the index of $d_j(k)$ is $2^{j-1}(2k+1) = 2^{j-1} + k2^j$, or

$$d_j(k) \;=\; \texttt{u[(1<<j)/2 + (1<<j)*k]}. \tag{5.73}$$

To invert `haarlift()`, we reverse the order and use the inverses of the steps:

Invert the Haar Filter Transform to $2q$ samples

```
ihaarlift( u[], q, dq ):
[1] For n=0 to q-1, replace u[(2*n)*dq] /= sqrt(2.0)
[2] For n=0 to q-1, replace u[(2*n+1)*dq] *= sqrt(2.0)
[3] For n=0 to q-1, replace u[(2*n)*dq] -= 0.5*u[(2*n+1)*dq]
[4] For n=0 to q-1, replace u[(2*n+1)*dq] +=  u[(2*n)*dq]
```

Similarly, we can compute the inverse DWT by reversing the order of lifting and recursion:

Inverse Discrete Haar Transform by Lifting on 2^J Samples

```
ildht0( u[], J, dq ):
[0] If J>0, then do [1] to [2]
[1]    Compute ildht0( u[], J-1, 2*dq )
[2]    Compute ihaarlift( u[], (1<<J)/2, dq )
```

It is an exercise to show that `ildht0()` inverts `ldht0()`. The proof does not use

the orthogonal CQF conditions, as these are not needed to prove that `ihaarlift()` inverts `haarlift()`. Lifting may thus be used to get invertible discrete wavelet transforms from nonorthogonal filter transforms.

Symmetric extension before periodization

Suppose $u = \{u(k) : 0 \le k < N\}$ is a finite sequence of N samples of a smooth function. Then $|u(n) - u(m)|$ will be small when $|n - m|$ is small. However, the periodic extension of u given by $u(k + N) \stackrel{\text{def}}{=} u(k)$ will not have that same smoothness property in general, unless $|u(0) - u(N-1)|$ is also small. Analysis methods which require that u is a smooth N-periodic function might therefore show artifacts from this failure of the hypothesis. In Chapter 3, we saw that *smooth local periodization* could be used to modify a smooth signal made nonsmooth by restriction to a segment, but that technique is not available if all we have are the N samples and nothing outside.

Another way to preserve some smoothness after periodizing is to extend u by reflection before periodizing. This creates two-fold redundancy, but if we then use a filter transform with the same reflection symmetry the redundancy can be removed. Restriction to half the output coefficients determines the other half by symmetry, so there is neither redundancy nor loss of information.

There are two ways to extend u symmetrically past $k = N - 1$ and $k = 0$. One is *half-sample symmetric periodization*:

$$HSPu(k) = \begin{cases} u(k), & \text{if } k = 0, \ldots, N-1 \pmod{2N}; \\ u(2N-1-k), & \text{if } k = N, \ldots, 2N-1 \pmod{2N}. \end{cases}$$

This extension has period $2N$ and satisfies $u(k) = u(-1-k) = u(2N-1-k)$ for all integers k, making it symmetric about the half-integers $-\frac{1}{2}$ and $N-\frac{1}{2}$. To compute $HSPu(k)$ at any $k \in \mathbf{Z}$ requires an index calculation:

Half-Sample Symmetric Periodic Extension

```
hsp( u[], k, N ):
[0] If k<0, then let k = -1-k
[1] Let k = k%(2*N)
[2] If k>N-1, then let k = 2*N-1-k
[3] Return u[k]
```

The Haar filter transform has half-sample symmetry. If we apply it to the $2N$-periodic signal $HSPu$ and restrict to one period, the first N outputs will be the same as the second N. Further restriction to the first half-period eliminates all redundancy. Note that N can be an odd number, generalizing `haarlift()` to signals u of arbitrary length. Implementation of this is left as an exercise.

Alternatively, we may perform *whole-sample symmetric periodization*

$$WSPu(k) = \begin{cases} u(k), & \text{if } k = 0, \ldots, N-1 \pmod{2N-2}; \\ u(2N-2-k), & \text{if } k = N, \ldots, 2N-3 \pmod{2N-2}. \end{cases}$$

coefficient	value	coefficient	value
$h(0)$	3/4	$g(-1)$	5/4
$h(\pm 1)$	1/2	$g(-1 \pm 1)$	−5/32
$h(\pm 2)$	1/8	$g(-1 \pm 2)$	−3/8
		$g(-1 \pm 3)$	−3/32

Table 5.2: Nonzero coefficients of the 4,2-biorthogonal filters.

This extension has period $2N-2$ and satisfies $u(k) = u(-k) = u(2N-2-k)$ for all integers k, making it symmetric about the integers 0 and $N-1$. To compute $WSPu(k)$ at any $k \in \mathbf{Z}$ requires another index calculation:

Whole-Sample Symmetric Periodic Extension

```
wsp( u[], k, N ):
[0] If k<0, then let k = -k
[1] Let k = k%(2*N-2)
[2] If k>N-1, then let k = 2*N-2-k
[3] Return u[k]
```

The 9,7- and 4,2-biorthogonal filter transforms, described below and in the exercises, both have whole-sample symmetry. If we apply one of them to the $(2N-2)$-periodic signal $WSPu$ and restrict to one period, the first N outputs will determine the second $N-2$. Once again, N can be an arbitrary odd or even length for u. The discrete wavelet transform thus obtained is free from the power-of-2 length condition.

Symmetric filters

A filter sequence f can have four types of symmetry:

Types of Symmetric Filters

WS *Whole-sample symmetry about C:* For some integer C, $f(k) = f(2C-k)$ for all integers k;

WA *Whole-sample antisymmetry about C:* For some integer C, $f(k) = -f(2C-k)$ for all integers k. This also implies that $f(C) = 0$;

HS *Half-sample symmetry about $C + \frac{1}{2}$:* For some integer C, $f(k) = f(2C+1-k)$ for all integers k;

HA *Half-sample antisymmetry about $C + \frac{1}{2}$:* For some integer C, $f(k) = -f(2C+1-k)$ for all integers k.

For Haar filters, h is HS about $\frac{1}{2}$ and g is HA about $\frac{1}{2}$.

The *4,2-biorthogonal filters* are WS about 0 and -1, respectively, as is evident from Table 5.2. All their other coefficients are zeroes.

coefficient	value	coefficient	value
$h(0)$	0.8526986790	$g(-1)$	0.7884856164
$h(\pm 1)$	0.3774028556	$g(-1 \pm 1)$	−0.4180922732
$h(\pm 2)$	−0.1106244044	$g(-1 \pm 2)$	−0.0406894176
$h(\pm 3)$	−0.0238494650	$g(-1 \pm 3)$	0.0645388826
$h(\pm 4)$	0.0378284555		

Table 5.3: Nonzero coefficients of the 9,7-biorthogonal filters.

Likewise, the *9,7-biorthogonal filters* h, g given approximately in Table 5.3 are WS about 0 and -1. All their other coefficients are zero. Exact formulas for the coefficients may be found in Daubechies on page 279, Table 8.3.

Note that neither pair h, g of WS filters are orthogonal CQFs, as they are not related by conjugacy. They nevertheless allow perfect reconstruction, as their combined action is invertible. The filter transform that inverts the h, g combined filter transform is given by their actual conjugates $\tilde{h}, \tilde{g}$, defined by

$$\tilde{h}(k) \stackrel{\text{def}}{=} (-1)^k g(-1-k); \qquad \tilde{g}(k) \stackrel{\text{def}}{=} (-1)^{k+1} h(-1-k). \tag{5.74}$$

The conjugate of a symmetric filter is also symmetric:

Lemma 5.14 *Suppose* $h = \{h(k)\}$ *is a filter sequence, and* $g = \{g(k)\}$ *is its conjugate defined by* $g(k) = (-1)^k h(2M-1-k)$*, then*

1. *h is WS about C if and only if g is WS about* $2M - C - 1$*;*
2. *h is WA about C if and only if g is WA about* $2M - C - 1$*;*
3. *h is HS about C if and only if g is HA about* $2M - C - 2 + \frac{1}{2}$*;*
4. *h is HA about C if and only if g is HS about* $2M - C - 2 + \frac{1}{2}$*.*

Proof: In the first case, compute

$$\begin{aligned} g(k) &= (-1)^k h(2M-1-k) = (-1)^k h(2C-2M+1+k) \\ &= (-1)^k h(2M-1-[4M-2C-2-k]) \\ &= (-1)^{4M-2C-2-k} h(2M-1-[4M-2C-2-k]) \\ &= g(4M-2C-2-k). \end{aligned}$$

Modified slightly, the same argument proves the second case:

$$g(k) = (-1)^k h(2M-1-k) = -(-1)^k h(2C-2M+1+k) = -g(4M-2C-2-k).$$

In the third case,

$$\begin{aligned} g(k) &= (-1)^k h(2M-1-k) = (-1)^k h(2C+1-2M+1+k) \\ &= (-1)^k h(2M-1-[4M-2C-3-k]) \\ &= -(-1)^{4M-2C-3-k} h(2M-1-[4M-2C-3-k]) \\ &= -g(4M-2C-3-k). \end{aligned}$$

In the fourth case,

$$\begin{aligned} g(k) &= (-1)^k h(2M-1-k) = -(-1)^k h(2C+1-2M+1+k) \\ &= -(-1)^k h(2M-1-[4M-2C-3-k]) \\ &= (-1)^{4M-2C-3-k} h(2M-1-[4M-2C-3-k]) \\ &= g(4M-2C-3-k). \end{aligned}$$

This also follows from the third case by exchanging h and g and then substituting $C \leftarrow 2M - C - 2 + \frac{1}{2}$. □

With symmetric filters and symmetric extension and periodization, a discrete wavelet transform may be performed on signals of arbitrary length.

We begin by separating the three kinds of lifting steps. The step that computes new values at odd multiples of the increment dq is called *prediction*:

Whole-Sample Symmetric Lifting: Prediction Step

```
wslpredict( u[], N, dq, coeff ):
[0] Let i = dq
[1] While i<N-2*dq, do [2] to [3]
[2]    Sum u[i] += coeff*(u[i-dq]+u[i+dq])
[3]    Increment i += 2*dq
[4] If i+dq<N, then sum u[i] += coeff*(u[i-dq]+u[i+dq])
[5] Else sum u[i] += 2*coeff*u[i-dq]
```

Step 4 handles the odd N/dq case. Step 5 handles even N/dq by whole-sample symmetric extension.

The step that computes new values at even multiples of dq is called *updating*:

Whole-Sample Symmetric Lifting: Updating Step

```
wslupdate( u[], N, dq, coeff ):
[0] Sum u[0] += 2*coeff*u[dq]
[1] Let i = 2*dq
[2] While i<N-2*dq, do [2] to [3]
[3]    Sum u[i] += coeff*(u[i-dq]+u[i+dq])
[4]    Increment i += 2*dq
[5] If i<N, then do [6] to [7]
[6]    If i+dq<N, then sum u[i] += coeff*(u[i-dq]+u[i+dq])
[7]    Else sum u[i] += 2*coeff*u[i-dq]
```

Steps 0 and 7 perform whole-sample symmetric extension. Step 7 is supplanted by step 6 if N/dq is even. Note that the inverse of `wslpredict(u,N,dq,coeff)` is `wslpredict(u,N,dq,-coeff)`, and the inverse of `wslupdate(u,N,dq,coeff)` is `wslupdate(u,N,dq,-coeff)`.

The third type of lifting step, normalization, does not require symmetry. It is always[6] performed jointly on both the even-indexed and odd-indexed elements:

Lifting: Normalization Step

```
lnormalize( u[], N, dq, coeff ):
[0] Let i = 0
[1] While i<N-2*dq, do [2] to [4]
[2]    Replace u[i] *= coeff
[3]    Replace u[i+dq] /= coeff
[4]    Increment i += 2*dq
[5] Replace u[i] *= coeff
[6] If i+dq<N, then replace u[i+dq] /= coeff
```

Step 6 handles the even N/dq case. The inverse of `lnormalize(u,N,dq,coeff)` is `lnormalize(u,N,dq,1/coeff)`.

The 4,2- and 9,7-biorthogonal filters of JPEG 2000

The so-called 4,2-biorthogonal discrete wavelet transform used in the lossless JPEG 2000 image compression algorithm is a whole-sample symmetric-extension algorithm implemented by lifting. The filter transform requires two predictions, one updating, and one normalization:

WS Lifting: 4,2-Biorthogonal Filter Transform

```
wsl42filter( u[], N, dq ):
[0] Compute wslpredict( u[], N, dq, alpha42 )
[1] Compute wslupdate( u[], N, dq, beta42 )
[2] Compute wslpredict( u[], N, dq, gamma42 )
[3] Compute lnormalize( u[], N, dq, zeta42 )
```

The lifting coefficients[7] are $\alpha_{42} = \frac{1}{4}$, $\beta_{42} = 1$, $\gamma_{42} = -\frac{3}{16}$, and $\zeta_{42} = \frac{1}{2}$. The inverse filter transform requires one normalization, one updating, and two predictions, with inverted coefficients:

WS Lifting: Inverse 4,2-Biorthogonal Filter Transform

```
wsl42ifilter( u[], N, dq ):
[0] Compute lnormalize( u[], N, dq, 1/zeta42 )
[1] Compute wslpredict( u[], N, dq, -gamma42 )
[2] Compute wslupdate( u[], N, dq, -beta42 )
[3] Compute wslpredict( u[], N, dq, -alpha42 )
```

[6]The HS/HA biorthogonal Haar transform on any number of points can be made orthogonal if the last output in the odd N/dq case is left unnormalized. See Exercise 20.

[7]Determining the lifting coefficients from the filter coefficients is beyond the scope of this text. See Daubechies and Swelden's article in the further readings for a complete derivation.

Following `ldht0()`, we can now implement the 4,2-biorthogonal wavelet transform recursively:

WS Lifting: 4,2-Biorthogonal Discrete Wavelet Transform

```
wsl42dwt( u[], N, dq, J ):
[0] If J>0, then do [1] to [2]
[1]    Compute wsl42filter( u[], N, dq )
[2]    Compute wsl42dwt( u[], N, 2*dq, J-1 )
```

Reconstruction from the output coefficients is accomplished by the inverse:

WS Lifting: 4,2-Biorthogonal Inverse Discrete Wavelet Transform

```
wsl42idwt( u[], N, dq, J ):
[0] If J>0, then do [1] to [2]
[1]    Compute wsl42idwt( u[], N, 2*dq, J-1 )
[2]    Compute wsl42ifilter( u[], N, dq )
```

The 9,7-biorthogonal filter transform used in JPEG 2000 is implemented by whole-sample symmetric lifting as well:

WS Lifting: 9,7-Biorthogonal Filter Transform

```
wsl97filter( u[], N, dq ):
[0] Compute wslpredict( u[], N, dq, alpha97 )
[1] Compute wslupdate( u[], N, dq, beta97 )
[2] Compute wslpredict( u[], N, dq, gamma97 )
[3] Compute wslupdate( u[], N, dq, delta97 )
[4] Compute lnormalize( u[], N, dq, zeta97 )
```

The lifting coefficients, at better than IEEE 64-bit precision, are

$$\begin{aligned}
\alpha_{97} &= -1.586134342059923558428315451337 4\ldots \\
\beta_{97} &= -0.052980118572961414624129567503 5\ldots \\
\gamma_{97} &= +0.882911075530933295919790099003\ldots \\
\delta_{97} &= +0.443506852043971152115604215168 9\ldots \\
\zeta_{97} &= +1.149604398860241159795075642191\ldots
\end{aligned} \tag{5.75}$$

The rest of the implementation is left as an exercise.

5.3 Exercises

1. Draw graphs like those in Figure 5.1 on page 135, depicting the configuration of $w(at)$ and $w(t-b)$ in Case 6 and Equation 5.3 on page 134. Use the graphs to compute the inner product integrals $\sqrt{a}\int \bar{w}(at)w(t-b)\,dt$ for those two cases.

2. Show that the function

$$\rho(a,b) \stackrel{\text{def}}{=} \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix}$$

is a faithful representation of the group **Aff**.

3. Suppose that $f = f(a,b)$ is a function on **Aff**. Along with left-invariant integrals, it is possible to define an area element $d\mathbf{b}$, for $\mathbf{b} = (a,b) \in \mathbf{Aff}$, so that for any fixed $\mathbf{b}' \in \mathbf{Aff}$,

$$\int_{\mathbf{Aff}} f(\mathbf{b})\, d\mathbf{b} = \int_{\mathbf{Aff}} f(\mathbf{b}\mathbf{b}')\, d\mathbf{b}.$$

In other words, the integral should not be changed by renaming the group elements through applying $\mathbf{b}'$ on the right. An area element $d\mathbf{b}$ with these properties is called a *right invariant measure*. Find a normalized right invariant measure on **Aff**.

4. Suppose that $w = w(t)$ is the Haar mother function and $u = u(t)$ is an absolutely integrable function which is piecewise constant on intervals of the form $[k, k+1)$, where k is an integer. Prove that

$$u(k) = \sum_{j=1}^{\infty} 2^{-j/2} Wu(2^j, k).$$

5. Let d be a positive integer. Show that the function

$$\phi(\xi) = \begin{cases} e^{-(\log|\xi|)^2}, & \text{if } \xi \neq 0; \\ 0, & \text{if } \xi = 0, \end{cases}$$

has d continuous derivatives.

6. Let n and d be positive integers. Show that the function $\phi = \phi(\xi)$ defined in Exercise 5 satisfies $\phi^{(n)}(\xi) = O(1/|\xi|^d)$ as $\xi \to \pm\infty$.

7. Let $u = u(x)$ be any fixed function in $L^2(\mathbf{R})$. Let $w = w(t)$ be the function defined by $\mathcal{F}w(\xi) = \phi(\xi)$ as in Exercise 5. Prove that the wavelet transform $Wu = Wu(a,b)$ using mother function w has derivatives with respect to a and b which are continuous away from $a = 0$.

8. Compute $\|w\|$, where $w(x) = \operatorname{sinc} x = \frac{\sin \pi x}{\pi x}$. (Hint: use Plancherel's theorem.)

9. Show that the function $w(x) = 2\operatorname{sinc} 2x - \operatorname{sinc} x = \frac{\sin 2\pi x - \sin \pi x}{\pi x}$ is an admissible function. Compute its normalization constant.

10. Suppose that $w = w(x)$ is the Haar mother function. Compute the Fourier integral transform of w.

11. Prove that the set of functions $\{\phi_k : k \in \mathbf{Z}\}$ defined by $\phi_k(t) = \operatorname{sinc}(t-k)$ is orthonormal. (Hint: use Plancherel's theorem and the fact that $\widehat{\operatorname{sinc}} = \chi_{[-\frac{1}{2},\frac{1}{2}]}$.)

12. Show that if $h = \{h(k) : k \in \mathbf{Z}\}$ is the low-pass filter from an MRA, and M is any fixed integer, then defining

$$g(k) = (-1)^k h(2M-1-k), \qquad \text{for all } k \in \mathbf{Z}, \tag{5.76}$$

gives another high-pass filter for that MRA.

13. Show that the Daubechies 4 low-pass filter of Equation 5.49 is derived from the general form in Equation 5.48 using the parameter value $c = 2 - \sqrt{3}$.

14. Suppose that h, g is a CQF pair derived from an orthogonal MRA with scaling function ϕ and mother function ψ. Suppose that $u \subset L^2(\mathbf{R})$ has compact support, and let $s_j(k) \stackrel{\text{def}}{=} \langle \phi_{jk}, u \rangle$ and $d_j(k) \stackrel{\text{def}}{=} \langle \psi_{jk}, u \rangle$ be the expansion coefficients of u in V_j and W_J in their respective orthonormal bases. Prove that

$$\begin{aligned}
s_{j+1}(n) &= \sum_k h(k) s_j(2n+k) &&= \sum_k h(k-2n) s_j(k); \\
d_{j+1}(n) &= \sum_k g(k) s_j(2n+k) &&= \sum_k g(k-2n) s_j(k); \\
s_{j-1}(n) &= \sum_k h(n-2k) s_j(k) + g(n-2k) d_j(k).
\end{aligned}$$

(These are Equations 5.58, 5.59, and 5.60, respectively.)

15. Suppose that x, y, a, b, c, d are integers with $x \geq y$, $b \geq a$, and $d \geq c$.

a. Show that $2\left\lceil \frac{x-b}{2} \right\rceil + a \leq x - (b-a-1)$, and $2\left\lfloor \frac{y-a}{2} \right\rfloor + b \geq y + (b-a-1)]$. (Hence a sequence u supported in $[x, y]$ may have a projection F^*Fu, as defined by Equation 5.63, with coefficients in the larger interval $[x - (b-a-1), y + (b-a-1)]$.)

b. Show that

$$\begin{aligned}
\left(1 + \left\lfloor \frac{y-a}{2} \right\rfloor - \left\lceil \frac{x-b}{2} \right\rceil\right) &+ \left(1 + \left\lfloor \frac{y-c}{2} \right\rfloor - \left\lceil \frac{x-d}{2} \right\rceil\right) \\
&\geq 1 + y - x + \frac{b-a-1}{2} + \frac{d-c-1}{2}.
\end{aligned}$$

(This estimate of the quantity in Equation 5.67 shows that the nonperiodic discrete wavelet transform with CQFs longer than 2 may have more output coefficients than inputs.)

16. Show that if $h = \{h(k) : k \in \mathbf{Z}\}$ and $g = \{g(k) : k \in \mathbf{Z}\}$ satisfy the orthogonal CQF conditions, and $P = 2P'$ is any fixed *even* integer, then the P-periodizations h_P, g_P of h and g, respectively, also satisfy the orthogonal CQF conditions. Namely, show:

Normalization of h_P: $\sum_{k=0}^{P'-1} h_P(2k) = \sum_{k=0}^{P'-1} h_P(2k+1) = 1/\sqrt{2}$, and thus $\sum_{k=0}^{P-1} h_P(k) = \sqrt{2}$.

Self-orthonormality of h_P: $\sum_{k=0}^{P-1} h_P(k+2n)h_P(k+2m) = \delta_{P'}(n-m)$, for all integers n, m.

Normalization of g_P: $\sum_{k=0}^{P'-1} g_P(2k) = -\sum_{k=0}^{P'-1} g_P(2k+1) = 1/\sqrt{2}$, and thus $\sum_{k=0}^{P-1} g_P(k) = 0$.

Self-orthonormality of g_P: $\sum_{k=0}^{P-1} g_P(k+2n)g_P(k+2m) = \delta_{P'}(n-m)$, for all integers n, m.

Periodic independence of h_P and g_P: $\sum_{k=0}^{P-1} g_P(k+2n)h_P(k+2m) = 0$ for all integers n, m.

Periodic completeness of h_P and g_P: for all integers n, m,

$$\sum_{k=0}^{P'-1} [h_P(2k+n)h_P(2k+m) + g_P(2k+n)g_P(2k+m)] = \delta_P(n-m).$$

17. Implement the inverse to Mallat's periodic discrete wavelet transform, for signals of period $N = 2^J K$ with positive integer J and K, using arbitrary 4-tap filters. Use it to generate a graph of the Daubechies 4 wavelet and scaling function, using the filters

$$\begin{aligned}
-g(3) = h(0) &= \frac{1+\sqrt{3}}{4\sqrt{2}} &\approx\ & 0.48296291314453416 \\
g(2) = h(1) &= \frac{3+\sqrt{3}}{4\sqrt{2}} &\approx\ & 0.83651630373780794 \\
-g(1) = h(2) &= \frac{3-\sqrt{3}}{4\sqrt{2}} &\approx\ & 0.22414386804201339 \\
g(0) = h(3) &= \frac{1-\sqrt{3}}{4\sqrt{2}} &\approx\ & -0.12940952255126037
\end{aligned}$$

18. Suppose that $\{u(n) : n \in \mathbf{Z}, 0 \le n \le 2q-1\}$ is a given sequence of $2q$ numbers.

a. Determine the contents of $u(2n)$ and $u(2n+1)$, for $n = 0, 1, \ldots, q-1$, after the following sequence of substitutions:

1. $u(2n+1) \leftarrow u(2n+1) - u(2n)$, all $n = 0, \ldots, q-1$;
2. $u(2n) \leftarrow u(2n) + \frac{1}{2}u(2n+1)$, all $n = 0, \ldots, q-1$;

3. $u(2n+1) \leftarrow u(2n+1)/\sqrt{2}$, all $n = 0, \ldots, q-1$;
4. $u(2n) \leftarrow \sqrt{2}\,u(2n)$, all $n = 0, \ldots, q-1$.

(These are the steps performed by `haarlift()`.)

b. Determine the contents of $u(2n)$ and $u(2n+1)$, for $n = 0, 1, \ldots, q-1$, after the following sequence of substitutions:

1'. $u(2n) \leftarrow u(2n)/\sqrt{2}$, all $n = 0, \ldots, q-1$;
2'. $u(2n+1) \leftarrow \sqrt{2}\,u(2n+1)$, all $n = 0, \ldots, q-1$;
3'. $u(2n) \leftarrow u(2n) - \frac{1}{2}u(2n+1)$, all $n = 0, \ldots, q-1$;
4'. $u(2n+1) \leftarrow u(2n+1) + u(2n)$, all $n = 0, \ldots, q-1$.

(These are the steps performed by `ihaarlift()`.)

c. Show that the procedure in part a is inverted by the procedure in part b, proving that `ihaarlift()` is the inverse of `haarlift()`.

d. Show that `ildht0()` is the inverse of `ildht0()`.

19. Implement the 9,7-biorthogonal discrete wavelet transform and inverse, for signals of arbitrary length, using whole-sample symmetric lifting and the coefficients of Equation 5.75. Plot a graph of a 100-point signal reconstructed from a single nonzero wavelet coefficient at index 32 at level 3. Do the same for index 36 at level 4, and again for index 41 at level 5.

20. Implement the discrete Haar wavelet transform and its inverse, for signals $u(0), \ldots, u(N-1)$ of arbitrary length N, using half-sample symmetric extension at $-\frac{1}{2}$ and $N - \frac{1}{2}$. Test your program by implementing the inverse to check that it recovers the signal. Modify the normalization step to omit normalizing $u(N-1)$ when N is odd, and test that the resulting transform is orthogonal by computing the sum-of-squares of signal samples and transform coefficients to check that they are equal.

21. Let $w = w(x)$ be the Haar mother function and let $\mathbf{1} = \mathbf{1}(x)$ be the Haar father function, namely the indicator function of $[0, 1]$. For $(x, y) \in \mathbf{R}^2$, define

$$e_0(x,y) = \mathbf{1}(x)\mathbf{1}(y), \qquad e_1(x,y) = \mathbf{1}(x)w(y)$$
$$e_2(x,y) = w(x)\mathbf{1}(y), \qquad e_3(x,y) = w(x)w(y).$$

Prove that the functions $\{e_n : n = 0, 1, 2, 3\}$ are orthonormal in the inner product space of square-integrable functions on $\mathbf{R}^2$.

5.4 Further Reading

- Alfred S. Cavaretta, Wolfgang Dahmen, and Charles A. Micchelli. *Stationary Subdivision*, volume 93(453) of the *Memoirs of the AMS*. American Mathematical Society, Providence, Rhode Island, 1991. ISBN 0-8218-2507-0.

- Ingrid Daubechies. *Ten Lectures on Wavelets*, volume 61 of the *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM Press, Philadelphia, 1992. ISBN 0-89871-274-2.

- Ingrid Daubechies and Wim Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.

- Arne Jensen and Anders la Cour-Harbo. *Ripples in Mathematics: The Discrete Wavelet Transform*. Springer-Verlag, Berlin, 2001. ISBN 3-540-41662-5.

- ISO/IEC. JPEG 2000 final committee draft 15444-1, March 2000. Available from URL `http://www.jpeg.org/JPEG2000.html`.

- Stéphane G. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, New York, second edition, 1999. ISBN 0-12-466606-X.

Chapter 6

Redundancy and Information

Information is stored in computers as strings of bits. For example, each letter, number and punctuation mark of English text in my computer is encoded with a unique string of seven bits defined by the *American Standard Code for Information Interchange*, or *ASCII*, a 128-character alphabet listed in Table B.1 on page 274 in Appendix B.

Today's most common computers use eight bits per character in a 256-element alphabet, because this simplifies and standardizes hardware designs. With ASCII data, the eighth bit can be used for some other purpose, such as parity-error detection, described in Section 6.2 further on. The eighth bit depends on the previous seven and is therefore redundant, adding to the cost of information storage and transmission.

There are subtler forms of redundancy as well. For example, the letter 'e' is exceedingly common in English text. The string for 'e' can be abbreviated by one bit if the string for a rare letter such as 'q' is lengthened by one bit. The result is a slightly shorter encoding of English text. How far can this idea be pushed?

Likewise, not every sequence of letters constitutes an English word. Given the side information that the communication is English text, the receiver can often deduce the entire word or phrase from just a portion of its letters. This is evident in expressions like '2 bdrm aptmnt avlbl'. With sufficient effort, all the possible interpretations can be searched and the likeliest one chosen. What is the cost to benefit relationship for such methods?

The theoretical lower bound on the amount of information needed to represent a string of bits is the *algorithmic complexity*, or *Kolmogorov complexity*, which is defined to be the number of bits needed to specify the shortest algorithm that produces the bit string and then terminates. Such a vague definition requires a notion of "universal computer," programmable by bit strings to produce any output bit sequence. Both receiver and transmitter must possess such a computer—it provides the context in which the algorithm is interpreted—but that is not an obstacle in practice. However, algorithmic complexity is an impractical information measure for two reasons. First, it assigns no weight to the effort required for

encoding or decoding. It is possible that the shortest algorithm will produce the output after a million times as many steps as an algorithm only one bit longer. A less dramatic example is the compressed message '2bdrmaptmntavlbl' which is only three space-characters shorter than the original but is much harder to decode. Second, the effort needed to find the shortest algorithm grows exponentially with the length of the message to be encoded. That is because in the most general case it is necessary to examine all of the algorithms with bit lengths up to that of the message, and there are 2^N of these for a message originally encoded with N bits. There can be no guarantee that every message will require a significantly shorter algorithm, for if so then a small number of algorithms could be used to denumerate a much larger number of messages. Because of these flaws, the notion of algorithmic complexity is mainly useful for obtaining ultimate lower bounds for data compression methods.

6.1 Information Source Coding

There is a more practical notion of complexity which, for certain kinds of messages, gives a lower bound on the average number of bits per transmitted character. We begin with rigorous definitions of some of the intuitive notions just mentioned:

Definition 4 *An* information source *is a random variable S taking values in the alphabet A.*

The *alphabet* is a finite set $A = \{a_1, \ldots, a_N\}$. We imagine evaluating S over and over, to get information in the form of a sequence of values. The probability space for S is the set A, and the N elements $a_1, \ldots, a_N$ of A are called *letters* or *characters*.

The probability function is completely determined by its values $p_k \stackrel{\text{def}}{=} \Pr(S = a_k)$ on individual letters. These are also called the *occurrence probabilities*. We have $0 \leq p_k \leq 1$ for all $k = 1, \ldots, N$, and $p_1 + \cdots + p_N = 1$.

A *message of length M* from the source S is the random variable $S_1 \cdots S_M$ built from M independent copies of S. Its probability space is

$$A^M \stackrel{\text{def}}{=} \overbrace{A \times \cdots \times A}^{M},$$

the ordered M-tuples of letters in A. A *message instance* is a particular sequence $s_1 \cdots s_M \in A^M$. It is written concatenated, without commas or parentheses. Independence implies that regardless of $m = 1, \ldots, M$,

$$\Pr(S_m = a_k) = p_k, \qquad \text{for each } k = 1, \ldots, N,$$

and thus for any message instance of M letters $s_1 \ldots s_M \in A^M$,

$$\Pr(S_1 \cdots S_M = s_1 \cdots s_M) \;=\; \Pr(S_1 = s_1) \times \cdots \times \Pr(S_M = s_M).$$

The assumption that adjacent letters in a real text are values of independent random variables is clearly false in particular cases. In English text, for example,

the likelihood that $S_{m+1} = u$ is strongly influenced by whether $S_m = q$. The independence assumption simply states that we will not take such influences into account when calculating the probability of a message instance. Our deliberate ignorance will produce cruder results, but they will apply to all languages.

6.1.1 Lossless encoding

Each character a in a message instance may be transmitted as a *codeword* $\mathbf{c}(a)$, a finite string of 1 and 0 bits[1]. The *code* $a \mapsto \mathbf{c}(a)$ is a function on the alphabet $A = \{a_1, \ldots, a_N\}$. We require that it be one-to-one, so that it can be inverted and the character recovered from its codeword. Since there are just 2^n possible n-bit codewords, it follows that if all the codewords have n or fewer bits, then $N \leq 2^n$.

If the codewords all have n bits and $\mathbf{c}$ is one-to-one, then every message instance $s_1 \cdots s_M$ for every M can be recovered exactly from its concatenated codeword bit strings $\mathbf{c}(s_1) \cdots \mathbf{c}(s_M)$, with no lost or extra letters. For example, ASCII encodes a 128-letter alphabet using $n = 7$ bits per codeword, the minimum number. But we will allow codewords of varying bit lengths, as long as the encoding function remains invertible:

Definition 5 *A code* $\mathbf{c}$ *for an alphabet* A *is called* uniquely decipherable *if the* message encoding function *that concatenates codewords,*

$$\mathbf{c}(s_1 s_2 \cdots s_M) \stackrel{\text{def}}{=} \mathbf{c}(s_1)\mathbf{c}(s_2) \cdots \mathbf{c}(s_M),$$

is one-to-one on A^M *for every* M.

One way to show that the message encoding function is one-to-one is to find an algorithm that recovers $s_1 \cdots s_M$ from its encoding $\mathbf{c}(s_1) \cdots \mathbf{c}(s_M)$. For example, $\mathbf{c}(a) = 0, \mathbf{c}(b) = 0001$ is a variable bit length code for $A = \{a, b\}$. The message instance *baab* yields the binary string 0001 0 0 0001. This code is uniquely decipherable, since the following algorithm recovers any message instance from its encoding:

Unique Deciphering Example

```
decipherex( bit[], L ):
[0] Let n=0
[1] For k=1 to L, do [2] to [5]
[2]    If bit[k]==0, then increment n += 1
[3]    Else do [4] to [5]
[4]       Print n-3 'a's followed by a single 'b'
[5]       Let n = 0
[6] Print n 'a's and terminate.
```

By contrast, $\mathbf{c}(a, b, c) = (0, 0, 1)$ is not a uniquely decipherable code for the alphabet $A = \{a, b, c\}$, since the codeword 0 corresponds to both a and b. The

[1]Only binary codes will be considered here. A more general treatment of codes in any base B can be found in Ash's *Information Theory*, pages 27-45.

problem is that its codewords are not long enough for it to be one-to-one. They must be long enough to distinguish different messages, giving the following necessary condition:

Lemma 6.1 *If alphabet A has a uniquely decipherable code with codeword lengths $\{n_x : x \in A\}$, then*

$$\sum_{x \in A} 2^{-n_x} \leq 1.$$

Proof: Since A is finite, there must be a longest codeword. Let $r > 0$ be its length in bits. Let w_j be the number of codewords of length j, for $j = 1, 2, \ldots, r$. Then

$$\sum_{x \in A} 2^{-n_x} = \sum_{j=1}^{r} w_j 2^{-j} \stackrel{\text{def}}{=} W.$$

Now consider W^m for any fixed $m \geq 1$. This consists of terms $w_{j_1} \cdots w_{j_m} 2^{-j}$, where $j = j_1 + \cdots + j_m$ for each j satisfying $m \leq j \leq mr$, so

$$W^m = \sum_{j=m}^{mr} \left(\sum_{j_1 + \cdots + j_m = j} w_{j_1} \cdots w_{j_m} \right) 2^{-j} \stackrel{\text{def}}{=} \sum_{j=m}^{mr} N_j 2^{-j}.$$

Now N_j counts all the m-letter messages whose encodings total j bits. Since each encoding is uniquely decipherable, N_j cannot exceed the number of j-bit binary strings, which is 2^j. Thus,

$$W^m = \sum_{j=m}^{mr} N_j 2^{-j} \leq \sum_{j=m}^{mr} 2^j 2^{-j} = \sum_{j=m}^{mr} 1 \leq mr,$$

so $W \leq m^{1/m} r^{1/m}$. Since this is true for any m, and $\lim_{m \to \infty} m^{1/m} r^{1/m} = 1$, it follows that $W \leq 1$. □

Note that if A has $N = 2^n$ letters, and each has a codeword of n bits as in ASCII where $n = 7$, then $\sum_{x \in A} 2^{-n_x} = N 2^{-n} = 1$ gives equality in Lemma 6.1. But equality is also attained by some variable-length uniquely decipherable codes. For example, $\mathbf{c}(a, b, c, d) = (0, 10, 110, 111)$ has codeword bit lengths $(n_a, n_b, n_c, n_d) = (1, 2, 3, 3)$, so $\sum_{x \in A} 2^{-n_x} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = 1$.

A code will also fail to be uniquely decipherable if two codewords can concatenate to give a third. For example, $\mathbf{c}(a, b, c) = (0, 01, 001)$ is not a uniquely decipherable code for the alphabet $A = \{a, b, c\}$, since both ab and c are encoded as the string 001. We can avoid that problem by using a fixed bit length for all codewords, or else by imposing a structure on the code:

Definition 6 *A* prefix code *is a uniquely decipherable code with the additional property that no codeword is the first part of a longer codeword.*

For example, $\mathbf{c}(a,b,c) = (0,11,101)$ is a prefix code, while $\mathbf{c}(a,b) = (0,0001)$, $\mathbf{c}(a,b,c) = (0,01,001)$ and $\mathbf{c}(a,b,c) = (0,0,1)$ are not.

Note that a prefix code is *instantaneous*, since each letter of the message can be determined as soon as its codeword is read. From our examples, we see that not all uniquely decipherable codes are prefix codes or instantaneous codes.

To describe the construction, encoding, and decoding of a prefix code, we first need some basic facts about abstract graphs and trees:

Definition 7 *A* graph *is a set V, called the* vertices, *together with a set E of (unordered) pairs of vertices called* edges.

- *A graph is called* finite *if there are finitely many vertices, which implies there are finitely many edges*[2]. *A graph with n vertices can have at most $n(n-1)/2$ edges.*
- *Two vertices $u, v \in V$ are considered* joined by an edge, *or* neighbors, *if $\{u,v\} \in E$.*
- *Two vertices $u, v \in V$ are considered* joined by a path *if there is a finite set of distinct vertices $\{u = x_0; x_1 \ldots, x_{n-1}; x_n = v\}$ such that $\{x_{k-1}, x_k\} \in E$ for every $k = 1, 2, \ldots, n$. Such a path is said to pass through the* intermediate vertices x_1, x_2, $\ldots$, x_{n-1}. *The* length *of this path is n, the total number of edges it contains.*
- *A graph is called* connected *if each pair of vertices is joined by a path.*
- *The* distance *between two distinct vertices is the length of the shortest path joining them. If they are not joined, then their distance is said to be $+\infty$. If two vertices are joined by an edge, their distance is 1. A vertex is also considered at zero distance from itself. In all three of these extreme cases, the set of intermediate vertices is empty.*

Graphs are often visualized as sets of points, the vertices, connected by line segments, the edges. Figure 6.1 shows two examples. It is left as an exercise to label the vertices and list the edges.

Definition 8 *A* tree *is a graph in which each pair of vertices is joined by a unique path.*

- *A tree contains no* loops, *or distinct paths between distinct vertices.*
- *A* leaf *is any vertex in a tree which is joined by an edge to a single other vertex.*
- *We can also fix one vertex in a tree and call it the* root. *Any vertex will do, but there can be no more than one. If a tree has a root, then*

[2]Our definition allows at most one edge for each pair of distinct vertices. What happens if we allow a vertex to be joined to itself by an edge, or allow more than one edge between two vertices?

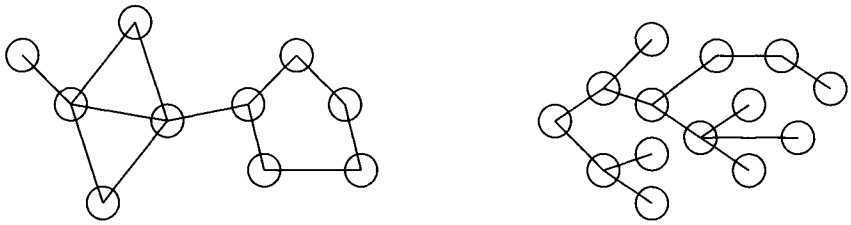


Figure 6.1: Example graphs: a generic graph with loops, and a tree.

- *The* generation *of a vertex in the tree is the distance between it and the root. The root is generation 0.*
- *Every non-root vertex is joined by an edge to a unique* parent *vertex one generation closer to the root. The root vertex has no parent.*
- *The* children *of any vertex are the vertices one generation farther from the root that are joined to it by an edge. A vertex with no children is a leaf.*
- Ancestors *of a vertex include the parent, the grandparent and so on;* descendents *include children, grandchildren, and so on.*
- *The* depth *of a tree with root is the maximum generation number of any vertex.*

The graph on the right-hand side of Figure 6.1 is a tree. It is left as an exercise to pick a root and find the resulting depth and the generation numbers of each of its vertices.

Trees with a root are used in the construction of uniquely decipherable codes, in fact prefix codes, because of their unique path property. The generation of a vertex will be the bit length of a codeword. For proper labeling, we must add one extra piece of structure to a tree:

Definition 9 *A* binary tree *is a tree with root such that each vertex has at most two children, which are distinguishable and called the* son *and* daughter.

This extra structure affects the enumeration of binary trees. For example, the graphs in Figure 6.2 are considered different as binary trees, even though they are identical as graphs and trees. The number B_n of binary trees with n vertices is[3]

$$B_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{n!(n+1)!}. \tag{6.1}$$

Still, for every n there are only finitely many binary trees with n vertices. There are likewise at most finitely many binary trees with D or fewer generations, since such a tree can have at most $n = 1 + 2 + \cdots + 2^D = 2^{D+1} - 1$ vertices. However, for every $N > 0$ there are infinitely many binary trees with N leaves.

[3]See Knuth, pages 388–389, from the further readings in Chapter 1.

Figure 6.2: Two equivalent graphs which are considered distinct binary trees.

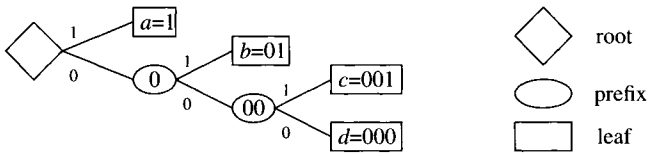


Figure 6.3: Binary tree for the prefix code $(a = 1, b = 01, c = 001, d = 000)$.

A special kind of binary tree is the *extended binary tree*, which has the property that every non-leaf, or *interior vertex*, has both a son and a daughter vertex. Any binary tree with n vertices can be grown into an extended binary tree by adding a son or daughter leaf wherever one or both are missing. The extended tree will have $n + 1$ added leaves and $2n + 1$ total vertices, with all of its original vertices becoming interior vertices. Thus an extended binary tree with N leaves has $N - 1$ interior vertices and $2N - 1$ total vertices.

A *quaternary tree* or *quadtree* is a tree with root such that each vertex has at most four distinguishable children. In general, a d-ary tree has a root and at most d distinguishable children per vertex. The number of d-ary trees with n vertices is

$$D_n = \frac{1}{[d-1]n+1}\binom{dn}{n} = \frac{(dn)!}{n!([d-1]n+1)!}. \tag{6.2}$$

Extended d-ary trees can likewise be defined.

Any binary prefix code is equivalent to a binary tree. Each letter corresponds to a leaf in the tree, with its codeword sequence of 0's and 1's determined by the sequence of sons and daughters needed to get from the root to the leaf. For example, the code $\mathbf{c}(a, b, c, d) = (1, 01, 001, 000)$ corresponds to the tree in Figure 6.3. The generation number of a leaf in such a tree equals the number of bits in its codeword. For example, letters c and d in Figure 6.3 come from generation 3 and have 3-bit codewords. The deciphering algorithm starts at the root and turns left or right depending upon the latest bit read, until it gets to a leaf. It then prints the letter determined by that leaf, returns to the root, and continues reading bits.

Whenever a uniquely decipherable code exists, there is a prefix code with the same codeword bit lengths. The proof is constructive, building the prefix code's binary tree:

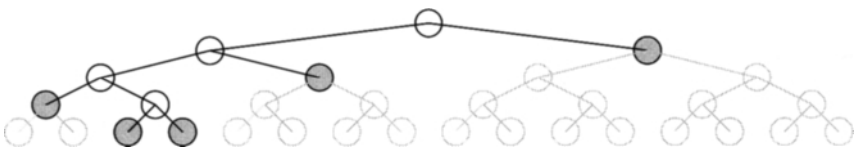


Figure 6.4: Prefix code built from a complete binary tree of depth $r = 4$. Shaded vertices indicate codeword leaves $\{000, 0010, 0011, 01, 1\}$; light vertices and edges have been pruned.

Lemma 6.2 *An alphabet A has a prefix code with codeword lengths $\{n_x : x \in A\}$ if and only if*

$$\sum_{x \in A} 2^{-n_x} \leq 1.$$

Proof: Since a prefix code is uniquely decipherable, Lemma 6.1 implies that A has a prefix code only if $\sum_{x \in A} 2^{-n_x} \leq 1$. Conversely, suppose the inequality holds. Let $r = \max\{n_x : x \in A\}$ be the the longest bit length and consider the complete binary tree with depth r, namely, one that has 2^r leaves at generation r from the root. We proceed as follows:

Construct a Prefix Code With Specified Codeword Lengths

PC-1. If there are no letters in alphabet A, then terminate. Otherwise, take $x \in A$ and let n_x be its codeword bit length.

PC-2. Find an unlabeled vertex at generation n_x which does not lie on the path from the root to any previously labeled codeword leaf. Label that vertex with the letter x.

PC-3. Prune the tree at the vertex labeled x by removing all of its descendents, making it a codeword leaf.

PC-4. Replace $A \leftarrow A \setminus \{x\}$ and go to step PC-1.

It remains to show that the algorithm does not run out of vertices in step PC-2 before it exhausts the finite alphabet A. But making a leaf labeled x at generation n_x removes exactly 2^{r-n_x} leaves from generation r. Since $\sum_{x \in A} 2^{-n_x} \leq 1$ implies that $\sum_{x \in A} 2^{r-n_x} \leq 2^r$, the tree provides at least enough unlabeled vertices for the whole alphabet. □

Figure 6.4 depicts the construction of a prefix code with lengths $3, 4, 4, 2, 1$ for a five-letter alphabet. Note that there may be more than one prefix code for a given set of bit lengths.

6.1.2 Efficient coding

All of the codes we consider are *static*, in the sense that each letter of the alphabet is assigned the same codeword regardless of its position in the message. That is

not a necessary assumption for uniquely decipherable codes, as one of the exercises will show, but it simplifies the analysis. In particular, it allows us to compute the following:

Definition 10 *The* bit rate *of a code* $\mathbf{c}$ *for an information source* S *is the expected number of bits per character in any message from* S.

Let $n(x_1 \cdots x_M)$ be number of bits in the encoding $\mathbf{c}(x_1 \cdots x_M)$ of the message instance $x_1 \cdots x_M \in A^M$ of length M, where each x_m is a letter in the alphabet A used by the source S. Since the code is static, $n(x_1 \cdots x_M) = n(x_1) + \cdots + n(x_M)$. Since the letters in a message from this source are independent, $\Pr(x_1 \cdots x_M) = \Pr(x_1) \cdots \Pr(x_M)$. The expected number of bits $R_M = R_M(S, \mathbf{c})$ in a message of length M is therefore

$$\begin{aligned} R_M &= \sum_{A^M} n(x_1 \cdots x_M) \Pr(x_1 \cdots x_M) \\ &= \sum_{x_1 \in A} \sum_{A^{M-1}} [n(x_1) + n(x_2 \cdots x_M)] \Pr(x_1) \Pr(x_2 \cdots x_M) \\ &= \left(\sum_{x_1 \in A} n(x_1) \Pr(x_1) \right) \left(\sum_{A^{M-1}} \Pr(x_2 \cdots x_M) \right) \\ &\quad + \left(\sum_{x_1 \in A} \Pr(x_1) \right) \left(\sum_{A^{M-1}} n(x_2 \cdots x_M) \Pr(x_2 \cdots x_M) \right). \end{aligned}$$

The sums without n are both 1, since Pr is a probability function, so we get a recursive formula $R_M = R_1 + R_{M-1}$, where

$$R_1 = \sum_{x \in A} n(x) \Pr(x) = \sum_{x \in A} n_x p_x. \tag{6.3}$$

Here n_x is the bit length of the codeword $\mathbf{c}(x)$, and p_x is the occurrence probability of letter x. But the recurrence is easy to solve by induction: $R_M = M R_1$, so the expected number of bits per character in a $\mathbf{c}$-coded message of any length is just R_1.

With our independence assumptions, the one and only way to reduce the bit rate is to match the bit lengths $\{n_x\}$ to the occurrence probabilities $\{p_x\}$. We will save bits on frequently occurring letters and spend them on rare characters, thereby reducing the average cost.

Lemma 6.5, proved below, shows that the following quantity is a sharp lower bound for this minimal average bit length per character:

$$H(p) \stackrel{\text{def}}{=} \sum_{x \in A} p_x \log_2 \frac{1}{p_x}, \tag{6.4}$$

where p_x is the occurrence probability of letter x. We evaluate $0 \log(1/0) \stackrel{\text{def}}{=} 0$ if $p_x = 0$. $H(p)$ is called the *entropy* of the message. It applies to all replacement

alphabets, so no matter how the original message is translated into codewords, the new version will still have the same occurrence probabilities $p = \{p_x : x \in A\}$ and thus the same entropy $H(p)$. Hence, entropy measures the minimum number of bits per character needed to distinguish among all messages with occurrence probabilities p, that is, the information contained in one such message.

Quantity 6.4 is also called the *first order entropy*, from the assumption that individual characters in the message are independent. *Second order entropy* counts the occurrence probability of pairs of characters, *third order entropy* that of triples, and so on. Higher-order entropies measure the information cost of messages wherein letters are partially predictable from their predecessors, namely for which the independence assumption fails.

Optimal coding efficiency

The expected number of bits per character needed by a uniquely decipherable (binary) code for a message in alphabet A is the sum

$$\sum_{x \in A} p_x n_x, \tag{6.5}$$

where p_x is the occurrence probability of letter x in the message, and n_x is the number of bits in the codeword for x. This is also called the *bit rate* of the code. The *efficient coding problem* for an alphabet A with probabilities $p = \{p_x : x \in A\}$ is the problem of finding a set of nonnegative integers $\{n_x : x \in A\}$ minimizing bit rate under the constraint

$$\sum_{x \in A} 2^{-n_x} \leq 1. \tag{6.6}$$

Lemma 6.3 *For any finite set of nonnegative probabilities $\{p_x : x \in A\}$, a solution exists to the efficient coding problem.*

Proof: Suppose that $A' \subset A$ indexes the strictly positive probabilities, with $\min\{p_x : x \in A'\} \stackrel{\text{def}}{=} p' > 0$ being the smallest. Any set $\{n_x : x \in A\}$ must satisfy

$$\sum_{x \in A} n_x p_x = \sum_{x \in A'} n_x p_x \stackrel{\text{def}}{=} \sum\nolimits' n_x p_x,$$

so it suffices to show that the latter sum over A' has a minimum that satisfies the constraint in Equation 6.6.

Put $d = \sum' 2^{-n_x}$. Since A' is finite, there is a bounded set of positive integers $\{n'_x : x \in A'\}$ for which $d < 1$. For example, it works to take $n'_x = n'$ for all $x \in A'$, where n' is the least integer greater than the base-2 logarithm of the number of elements of A. Define $d' = \sum' n'_x p_x$ to be the bit rate of this starting set of codeword lengths.

If $n_x > d'/p'$ for all $x \in A'$, then $\sum' n_x p_x > d'$ will exceed the starting set bit rate. Thus the minimum of $\sum' n_x p_x$ lies in the finite collection of sets $\{n_x : x \in A'\}$

for which $0 \le n_x \le d'/p'$ at every $x \in A'$. It can be found by searching the list and checking the constraint.

If there are no zero probabilities, the solution is the minimizing set that satisfies $\sum 2^{-n_x} \le 1$. If some of the probabilities are zero, the solution is the minimizer that satisfies $\sum' 2^{-n_x} < 1$, taking large enough integers n_x for all the irrelevant $x \notin A'$ to get $\sum 2^{-n_x} \le 1$. □

Once it is known that a solution exists, efficient algorithms may be used to find it. Lemma 6.2 implies that if A has a uniquely decipherable code with given codeword lengths, it also has a prefix code with the same set of codeword lengths. Hence, a prefix code for A can always be found which is just as efficient as any uniquely decipherable code for A. The optimization problem for prefix codes is easier to solve because of their correspondence with binary trees.

Before constructing a solution to the efficient coding problem, we show that the entropy $H(p)$, defined in Equation 6.4, gives a sharp lower bound for the bit rate of Equation 6.5. We start with an inequality provable by elementary calculus:

Lemma 6.4 *Suppose* $p = \{p_x : x \in A\}$ *and* $q = \{q_x : x \in A\}$ *satisfy*

$$0 < p_x, q_x \le 1 \qquad \text{and} \qquad \sum_{x \in A} p_x = \sum_{x \in A} q_x = 1.$$

Namely, p and q are two occurrence probabilities for an alphabet A. Then

$$\sum_{x \in A} p_x \log\left(\frac{1}{p_x}\right) \le \sum_{x \in A} p_x \log\left(\frac{1}{q_x}\right),$$

with equality if and only if $p_x = q_x$ for all $x \in A$.

Proof: The graph of $\log t$ lies below its tangent line at $t = 1$, so $\log t \le t - 1$, with equality if and only if $t = 1$. But then $\log(q_x/p_x) \le (q_x/p_x) - 1$ with equality if and only if $p_x = q_x$. Multiplying by p_x and summing over $x \in A$ yields

$$\sum_{x \in A} p_x \log \frac{q_x}{p_x} \le \sum_{x \in A} (q_x - p_x) = 0, \qquad \Rightarrow \sum_{x \in A} p_x \log \frac{1}{p_x} \le \sum_{x \in A} p_x \log \frac{1}{q_x},$$

with equality if and only if $p_x = q_x$ for all $x \in A$. □

Since $\log_2 x = (\log x)/(\log 2)$ for every $x > 0$, Lemma 6.4 holds for $\log_2$ as well: we just divide the inequality by the positive constant $\log 2$. That imposes a lower bound for the average number of bits per character of any prefix code:

Lemma 6.5 *For any prefix code with codeword lengths* $\{n_x : x \in A\}$,

$$\sum_{x \in A} p_x n_x \ge H(p).$$

Equality holds if and only if $p_x = 2^{-n_x}$ for each $x \in A$.

Proof: Put $W = \sum_{x\in A} 2^{-n_x}$, and define $q_x = 2^{-n_x}/W$ for each $x \in A$. Then $\sum_{x\in A} q_x = 1$. By Lemma 6.4,

$$H(p) = \sum_{x\in A} p_x \log_2 \frac{1}{p_x} \le \sum_{x\in A} p_x \log_2 \frac{1}{q_x},$$

with equality if and only if $q_x = p_x$ for all $x \in A$. Continuing the calculation,

$$\sum_{x\in A} p_x \log_2 \frac{1}{q_x} = \sum_{x\in A} p_x n_x + \log_2 W \sum_{x\in A} p_x \le \sum_{x\in A} p_x n_x,$$

since $W \le 1$ by Lemma 6.2, so $\log_2 W \le 0$. Equality holds here if and only if $W = 1$, so it holds throughout if and only if $p_x = q_x = 2^{-n_x}$ for all $x \in A$. □

Existence of almost optimal codes

A second important preliminary observation is that a prefix code can get arbitrarily close to the lower bound in Lemma 6.5. What makes this tricky is that a binary prefix code must use at least one bit per letter, while $H(p)$ can be less than one. This obstacle results in the following weak upper bound:

Lemma 6.6 *There exists a prefix code with codeword lengths* $\{n_x : x \in A\}$ *such that*

$$\sum_{x\in A} p_x n_x \le H(p) + 1.$$

Proof: Let $A' \subset A$ be the letters x with $p_x > 0$, and let $A'' \subset A$ be those with $p_x = 0$. Then $A = A' \cup A''$. For each $x \in A'$, let n_x be the unique integer satisfying $\log_2(1/p_x) < n_x \le 1 + \log_2(1/p_x)$. Then,

$$\sum_{x\in A'} 2^{-n_x} < \sum_{x\in A'} 2^{-\log_2(1/p_x)} = \sum_{x\in A'} p_x = \sum_{x\in A} p_x = 1.$$

For each $x \in A''$, we may take n_x large enough so that

$$\sum_{x\in A''} 2^{-n_x} \le 1 - \sum_{x\in A'} 2^{-n_x} \quad \Rightarrow \quad \sum_{x\in A} 2^{-n_x} \le 1.$$

By Lemma 6.2, there is a prefix code with codeword lengths $\{n_x : x \in A\}$. But

$$\sum_{x\in A} p_x n_x = \sum_{x\in A'} p_x n_x \le \sum_{x\in A'} p_x \left(1 + \log_2 \frac{1}{p_x}\right) = 1 + H(p),$$

so this prefix code has a bit rate below the claimed estimate. □

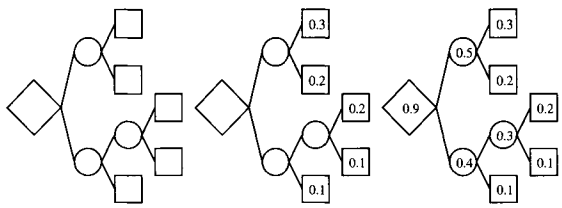


Figure 6.5: Propagating leaf weights back to the root of a binary tree.

Unfortunately, the upper bound of Lemma 6.6 is sharp. The trivial example $A = \{a\}$, with $p_a = 1$, has $H(p) = 0$, but we must have $n_a \geq 1$ so that $p_a n_a \geq 1 = H(p) + 1$. At least one bit must be transmitted per letter to tell the receiver that something is there.

One extra bit per character is a heavy price, but it can be reduced. The trick is to code multiple letters so that the excess per character is made arbitrarily small. Consider an indefinitely long message in which successive characters are independent and identically distributed random variables taking values in the alphabet A with occurrence probabilities $p = \{p(x) : x \in A\}$. Then the probability of a particular sequence $x_1 x_2 \cdots x_s$ of s characters appearing in the message is $p(x_1 x_2 \cdots x_s) = p(x_1)p(x_2) \cdots p(x_s)$. It is a straightforward exercise to prove that if $p^s = \{p(x_1 x_2 \cdots x_s) : (x_1, x_2, \ldots, x_s) \in A^s\}$, then $H(p^s) = sH(p)$, and that if a code uses N bits on average per block of s letters, then it uses N/s bits per letter on average. Lemma 6.6 implies that there is a prefix code for blocks of s characters satisfying $N \leq H(p^s) + 1 = sH(p) + 1$, so $N/s \leq H(p) + 1/s$. Such an argument proves the following result:

Lemma 6.7 *Let $\epsilon > 0$ be given, and let $s > 1/\epsilon$ be an integer. Then there is a prefix code for blocks of s characters which uses fewer than $H(p) + \epsilon$ bits per character.* □

6.1.3 Huffman's algorithm

Bit rate may be abstracted into *weighted depth* $\sum_{x \in A} p_x n_x$, defined for any tree with a root and with leaves $x \in A$ each having a generation number n_x and a nonnegative *weight* p_x. Weighted depth gives the bit rate of a code if the weights $\{p_x : x \in A\}$ are normalized like occurrence probabilities, with $\sum_{x \in A} p_x = 1$. We will construct a minimal bit rate code by building a binary tree of minimal weighted depth, using D. A. Huffman's *weight propagation* algorithm. Leaf weights are propagated back to the root as each interior vertex gets the sum of its children's weights, starting with the deepest non-leaf vertices. Figure 6.5 shows one example. We prove by induction on the depth of the tree that weight propagation gives each vertex the sum of the weights of its descendent leaves.

Weight propagation is equivalent to filling each non-leaf vertex along the unique path from the root to a leaf with the leaf's weight, then *superposing* or adding

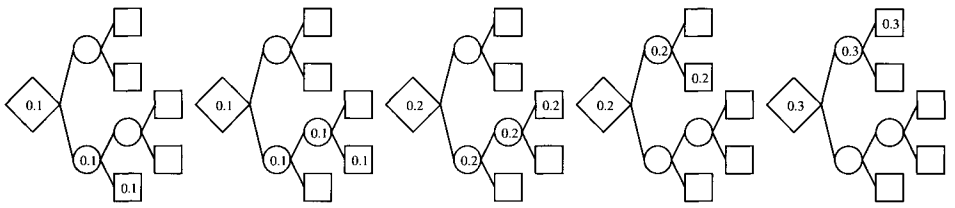


Figure 6.6: Binary trees weighted along root-to-leaf paths.

together all the weights. The resulting weight at each vertex is evidently the sum of its leaf descendent's weights. For example, Figure 6.6 shows five partially filled copies of the same tree which may be superposed to give the propagated tree of Figure 6.5.

In the superposed tree, a weight p_x from a leaf x at generation n_x appears $1+n_x$ times, so adding up the weights at all the vertices gives

$$\sum_{x \in A} p_x(1 + n_x) = \sum_{x \in A} p_x + \sum_{x \in A} p_x n_x.$$

But $\sum_{x \in A} p_x$ is the weight at the root after propagation, so we have:

Lemma 6.8 *The weighted depth of a binary tree equals the sum of the weights at all vertices except the root, after propagation.* □

Requiring minimal weighted depth constrains the shape of a binary tree:

Lemma 6.9 *Suppose that the leaf weights $p_1, \ldots, p_N$ are nonnegative. Then a binary tree of minimal weighted depth with these weights at its N leaves must have both a son and a daughter at every non-leaf vertex.*

Proof: Any non-leaf vertex with just one child may be removed and replaced by the child, shortening some paths and creating a new binary tree with smaller weighted depth. □

Lemma 6.9 implies that the minimal weighted depth tree must be an extended binary tree with N leaves. These are themselves binary trees with $2N+1$ vertices, and there are fewer than $\frac{(4N+2)!}{(2N+1)!(2N+2)!}$ of them by Equation 6.1. Hence to find the most efficient prefix code is at worst a search for minimal weighted depth through this many trees. However, Huffman found a more efficient method based on the following observation:

Lemma 6.10 *Given leaf weights $0 \leq p_1 \leq p_2 \leq \cdots \leq p_N$, there is a minimal weighted depth binary tree in which the lightest leaves, those weighted p_1 and p_2, share a parent.*

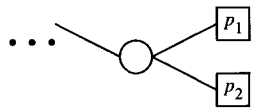


Figure 6.7: A minimal weighted depth tree whose lightest leaves are siblings.

Proof: Lemma 6.3 insures that at least one minimal weighted depth tree exists with the given leaf weights. We show that it or another minimal tree must contain the subtree depicted in Figure 6.7. Consider a non-leaf vertex with deepest generation number in a binary tree of minimal weighted depth. Its two children must be leaf vertices of maximal depth, so exchanging their weights with p_1 and p_2, which are lightest, does not increase the weighted depth, and gives a minimal weighted depth tree with the required subtree. □

A *Huffman code* is a prefix code for $N > 1$ letters with occurrence probabilities $p_1, p_2, \ldots, p_N$. *Huffman's algorithm* builds a binary *Huffman tree* from its N leaf vertices, each of which is assigned a weight p_k. To construct the tree, the parent of each vertex is specified recursively, beginning with the leaves:

Huffman's Algorithm, Recursive Version

HR-1. Sort and relabel the weights so that $0 \leq p_1 \leq p_2 \leq \cdots \leq p_N$.

HR-2. If $N = 2$, return a 3-vertex tree with root x_0 weighted $p_1 + p_2$, a son leaf x_1 weighted p_1, and a daughter leaf x_2 weighted p_2.

HR-3. Otherwise, construct a Huffman tree with the $N - 1$ leaf weights $(p_1 + p_2)$, p_3, ..., p_N. Then make the leaf weighted $(p_1 + p_2)$ into a parent interior vertex by appending a son leaf weighted p_1 and a daughter leaf weighted p_2. Return the appended tree.

Recursion step HR-3 is encountered at most $N - 2$ times.

Equivalently, the Huffman tree may be constructed with a loop. Until its parent is determined, call a vertex an *orphan*. We start by creating N orphan vertices with weights $p_1, \ldots, p_N$. Then we proceed through the following:

Huffman's Algorithm, Loop Version

HL-1. Find the orphan vertex x with the smallest weight p_x. Use some convention to break ties.

HL-2. Find the orphan vertex y with the next smallest weight p_y, again breaking ties by some convention. It is possible that $p_x = p_y$, but of course $x \neq y$.

HL-3. Create a new orphan vertex z, give it weight $p_z = p_x + p_y$, and make z the common parent of vertices x and y which are no longer orphans.

HL-4. If there is only one orphan vertex left, then call it the root and terminate. Otherwise, return to step HL-1.

$x \in A$	$p(x)$	$C(x)$
a	$1/2 = 0.5 = 0.100$ (base 2)	1
b	$1/4 = 0.25 = 0.010$ (base 2)	01
c	$1/8 = 0.125 = 0.001$ (base 2)	001
d	$1/8 = 0.125 = 0.001$ (base 2)	000

Table 6.1: Example Huffman code C for a four-letter alphabet.

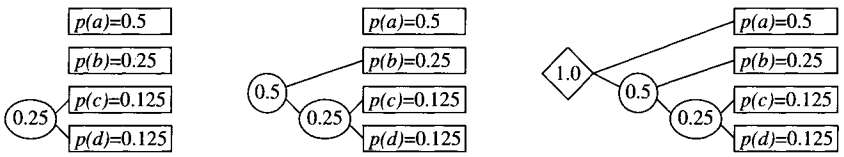


Figure 6.8: Three steps in building a Huffman tree.

For example, let $A = \{a, b, c, d\}$ be a four-letter alphabet. Table 6.1 shows one Huffman code chosen for a particular message written in this alphabet with occurrence probabilities $p(a, b, c, d) = (0.5, 0.25, 0.125, 0.125)$. Figure 6.8 shows the three passes through the loop that built the Huffman tree.

Notice that the code of Table 6.1 achieves the the entropy of the occurrence probabilities p, the minimum expected number of bits per letter for the message:

$$H(p) = \sum_{x \in A} p(x) \log_2 \frac{1}{p(x)} = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1.75$$

That is because the codeword bit lengths match the occurrence probabilities. It is not always possible to achieve the minimum, though, since codewords must have a positive integer number of bits whereas the probabilities need not be integer powers of 2.

Note too that Huffman codes are not uniquely determined. For the example code, exchanging the codewords for c and d gives another code. Indeed, exchanging 1 and 0 in each codeword of any binary code gives another code with the same bit rate.

Analysis of Huffman's algorithm

With each pass through the loop, or each recursion, the number of orphans and weights decreases by one, so either algorithm must terminate after $N-1$ steps. The sole remaining orphan will be the root and will have weight 1, since it accumulates the sum of all the probabilities $p_1 + \ldots + p_N = 1$.

Each reduction requires finding the two smallest weights among the remaining orphans. If there are k orphans, that costs $O(2k)$ comparison operations, for $k = N, N-1, \ldots, 2$. Adding up shows that Huffman's algorithm with N weights requires no more than $O(N^2)$ operations. It remains to show:

Theorem 6.11 *A Huffman tree has minimal weighted depth.*

Proof: Suppose that A has $N \geq 2$ letters with occurrence probabilities $p_1 \leq p_2 \leq \cdots \leq p_N$. The proof is by induction on N, using the recursive version of Huffman's algorithm.

If $N = 2$, step HR-2 returns the 3-vertex Huffman tree with a root and two leaves. It evidently has the minimal weighted depth $1 = 1 \cdot p_1 + 1 \cdot p_2$.

Now fix $N > 2$ and suppose that Huffman's algorithm returns a minimal weighted depth tree for any list of $N - 1$ weights. Step HR-3 of Huffman's algorithm constructs such an $(N - 1)$-leaf tree with leaf weights $(p_1 + p_2)$, p_3, $\ldots$, p_N, Denote its minimal weighted depth by e. Step HR-3 then appends two leaves, weighted p_1 and p_2, respectively, as son and daughter to the leaf weighted $(p_1 + p_2)$, and returns the resulting N-leaf tree. By Lemma 6.8, the weighted depth E of the Huffman tree returned from HR-3 is $E = e + p_1 + p_2$.

Now suppose that E' is the minimum weighted depth for the weights $p_1, \ldots, p_N$. By Lemma 6.10, there is a tree with this weighted depth in which the lightest leaves, weighted p_1 and p_2 because of HR-1, are siblings. Pruning these from their joint parent yields a tree for the $N - 1$ weights $(p_1 + p_2)$, p_3, $\ldots$, p_N with weighted depth $e' = E' - p_1 - p_2$, again computed by Lemma 6.8. Since $E' \leq E$ and $e \leq e'$, it follows that $E = E'$. Hence the returned Huffman tree for the N weights has minimal weighted depth. □

Describing a Huffman tree

To transliterate a message instance from alphabet A into its Huffman code, simply replace each letter with its binary codeword. This can be done with a table such as Table 6.1. No commas or other spacers are needed between codewords. However, to recover the message instance from the binary string of concatenated codewords, it is necessary to have the Huffman tree. The encoder that built the tree must transmit it to the receiver for use in decoding, and it should do so economically lest the advantage of Huffman coding be squandered in the cost of describing the tree.

One way to transmit the tree is to send the occurrence probabilities tagged with the letters they refer to, then have the receiver build the tree using exactly the same algorithm as the transmitter. For an N-letter alphabet, this side information costs $(B + \lceil \log_2 N \rceil) \times N$ bits, if we record the probabilities in a B-bit format. Probabilities are typically floating-point numbers, so B might be the rather costly 32 or 64. Also, to get exactly the same Huffman tree in all cases we must use exactly the same arithmetic to compare and add the weights. But arithmetic is not guaranteed to be the same on different computers. It is therefore better to describe a Huffman tree without mentioning the weights that built it.

We are aided by Lemma 6.9. A Huffman tree must be an extended binary tree, so with N leaves it will have $2N - 1$ total vertices and depth at most $N - 1$.[4]

[4]The depth of a Huffman tree is also bounded by $O(\log[p_N/p_1])$, where p_N and p_1 are the largest and smallest nonzero occurrence probabilities, respectively. In practice this is often much smaller than $N - 1$.

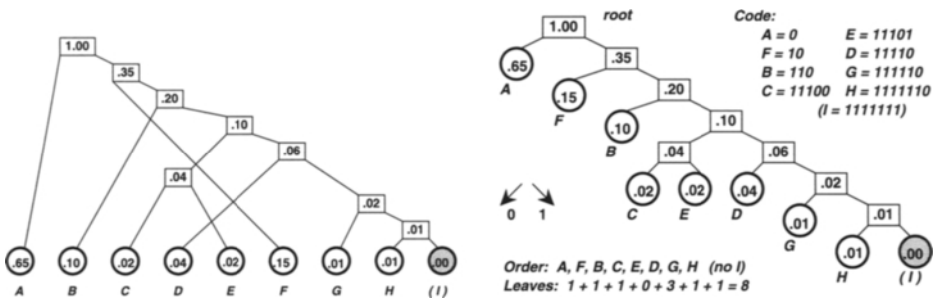


Figure 6.9: Example Huffman trees for a given alphabet and occurrence probabilities. Left: One possibility. Right: Canonical tree, after rearrangement.

The maximum number L of bits in a Huffman codeword is actually the generation number of the leaf with smallest positive weight, since deeper leaves with zero weight give *inactive* codewords that will never be transmitted. A Huffman tree thus determines an ordered list $M = (m_1, m_2, \ldots, m_L)$ of L small nonnegative integers. No integer in this list can be larger than N, the number of letters in the original alphabet. The element m_k is both the number of leaf vertices at generation k and the number of active codewords of length k. For the tree of depth $L = 3$ in Figure 6.3, we have $M = (1, 1, 2)$.

Since many equivalent trees will give the same list M, we will choose one of them to be the *canonical* tree. Our choice is the tree built from a rearrangement of occurrence probabilities such that, at each prefix node, the deeper subtree lies on the right. Figure 6.9 shows one such rearrangement. This convention fixes the sequence of 0s (left turns) and 1s (right turns) needed to go from the root to any leaf. Thus there is one and only one canonical Huffman tree for a given L and M.

Even given L and M, the Huffman code is not determined. Any permutation of the codewords of equal length k gives a code with the same bit rate, so we must also specify how leaves relate to letters. We thus provide an ordered list A' of the active letters of the alphabet A, giving the order in which they appear in the canonical tree, reading from left to right in increasing depth order. Thus letters a'_m, for $m = m_k + 1, \ldots, m_{k+1}$, are assigned to the leaves at generation number k in left-to-right order in the canonical tree. If $m_k = 0$, there are no codewords of length k.

These three data L, M, A' completely describe the canonical tree and are sufficient to encode and decode all the *active* letters of the alphabet, that is, those whose occurrence probabilities are nonzero. For example, the canonical tree on the right side of Figure 6.9 is uniquely described by $L = 7$, $M = (1, 1, 1, 0, 3, 1, 1)$, and $A' = a, f, b, , ced, g, h$. Commas in A' and M correspond.

A useful trick is to append an extra letter to the alphabet, with occurrence probability zero. This will guarantee that no active codeword consists of all 1 bits. It makes the decoding algorithm more robust, as the bit string of codewords can

be end-padded with 1 bits that cannot be confused with extra letters. An inactive letter adds no cost to the canonical Huffman tree description, but it does make the resulting code less efficient by an amount equal to the smallest nonzero occurrence probability. The letter i plays this special role in Figure 6.9, since its occurrence probability is 0, and it is not listed in A'.

6.2 Error Correction and Detection

Suppose that a bit in a stored or transmitted binary string flips to the other value: from 0 to 1 or from 1 to 0. This may result from some physical process such as a disk crash or line noise. A fixed-length encoding with a single flipped bit will decode into a message instance containing a single incorrect letter, with all the letters before and after it being correct. The shorter, variable-length Huffman encoding of the same message instance might suffer much greater damage. The flipped bit may divert the decoder to a deeper or shallower subtree, so that the wrong codeword and length will be read. The decoder will then try to decode the next letter starting in the middle of a codeword. The misalignment could even propagate so that all letters after the bit flip will be misread.

We are willing to pay some cost in extra bits to detect bit flips, as long as it does not negate the savings from redundancy removal. We will see that with a very few extra bits we may buy much more confidence in the correctness of a bit string. In some cases, a few extra bits even buy us the location of bit flips and allow us to correct them.

6.2.1 Parity bits

Extra bits can be added to a codeword bit string to detect errors. If seven bits within an eight-bit byte define an ASCII character, the extra bit can be set to zero if there are evenly many ones among the first seven bits, or set to one otherwise. Then the total number of one-bits in the string will always be even. Transmitter and receiver can agree to use this *even parity* condition to increase confidence in the correctness of the received data.

How much confidence is gained? To analyze this procedure in somewhat more generality, suppose that a bit string consists of N independent data bits plus one parity bit. Let p be the probability that a bit is flipped to the wrong value during transmission. Then the probability that exactly k data bits are flipped is $\binom{N}{k}p^k(1-p)^{N-k}$. It matters whether k is even or odd, and if even, whether k is zero, so we define the sets $E = \{0, 2, 4, \ldots\}$ and $O = \{1, 3, 5, \ldots\}$ of even and odd nonnegative integers, and put $E^+ = E \setminus \{0\} = \{2, 4, 6, \ldots\}$.

The receiver gets $N + 1$ bits, and there are four possible outcomes:

All's well. The parity is consistent and the data bits are all correct. Probability: $P_{aw} = (1-p)^{N+1}$.

False alarm. The parity is inconsistent but the data bits are all correct. Probability: $P_{fa} = p(1-p)^N$.

Detected error. The parity is inconsistent and at least one data bit is incorrect. Probability:

$$P_{de} = (1-p)\sum_{k\in O}\binom{N}{k}p^k(1-p)^{N-k} + p\sum_{k\in E+}\binom{N}{k}p^k(1-p)^{N-k}.$$

Undetected error. The parity is consistent but at least one data bit is incorrect. Probability:

$$P_{ue} = p\sum_{k\in O}\binom{N}{k}p^k(1-p)^{N-k} + (1-p)\sum_{k\in E+}\binom{N}{k}p^k(1-p)^{N-k}.$$

The sums may be evaluated by algebra. Note that

$$\sum_{k}\binom{N}{k}p^k(1-p)^{N-k} = (p+1-p)^N = 1$$

We may break the sum into even-index and odd-index parts:

$$x \stackrel{\text{def}}{=} \sum_{k\in E}\binom{N}{k}p^k(1-p)^{N-k}; \quad 1-x = \sum_{k\in O}\binom{N}{k}p^k(1-p)^{N-k}. \tag{6.7}$$

But then,

$$\begin{aligned} x-(1-x) &= \sum_{k\in E}\binom{N}{k}p^k(1-p)^{N-k} - \sum_{k\in O}\binom{N}{k}p^k(1-p)^{N-k} \\ &= \sum_{k\in E}\binom{N}{k}(-p)^k(1-p)^{N-k} + \sum_{k\in O}\binom{N}{k}(-p)^k(1-p)^{N-k} \\ &= \sum_{k}\binom{N}{k}(-p)^k(1-p)^{N-k} \qquad = (1-2p)^N. \end{aligned}$$

Solving for x gives

$$x = \frac{1+(1-2p)^N}{2}; \qquad 1-x = \frac{1-(1-2p)^N}{2}.$$

Thus, $P_{ue} = \frac{1}{2}[1+(1-2p)^{N+1} - 2(1-p)^{N+1}]$ and the probability that the data bits are all correct given that the parity is correct is

$$\frac{P_{aw}}{P_{aw}+P_{ue}} = \frac{2(1-p)^{N+1}}{1+(1-2p)^{N+1}}.$$

Likewise, $P_{de} = \frac{1}{2}[1-(1-2p)^{N+1} - 2p(1-p)^N]$ and the probability that all of the data bits are actually correct given that the received codeword has bad parity is

$$\frac{P_{fa}}{P_{de}+P_{fa}} = \frac{2p(1-p)^N}{1-(1-2p)^{N+1}}.$$

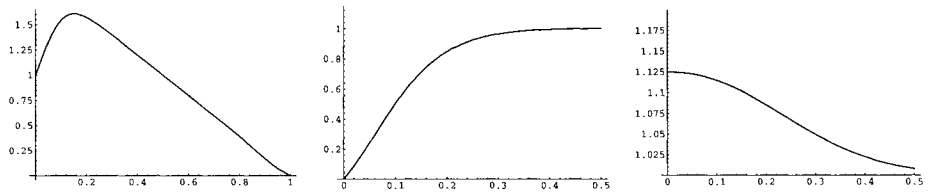


Figure 6.10: Parity checking performance with 7 data bits, as a function of bit flip probability p. Left: Data confidence increase. Center: Undetected error rate reduction. Right: False alarm rate increase.

The blind probability that all N data bits are correct is $(1-p)^N$. By accepting only strings with consistent parity, our confidence in the data's correctness is increased by a factor of $2(1-p)/(1+(1-2p)^{N+1})$, which is plotted for $N=7$ and $0<p<1$ at left in Figure 6.10. In this case, confidence is increased by the largest factor for bit flip probabilities near a decidedly unsatisfactory 0.2. When $p>0.5$, parity checking actually reduces confidence in the data!

In the limit $p\to 0$, where the data is pretty reliable even without parity checking, there is not much room for increased confidence, so error-rate reduction gives a more meaningful figure of merit. Parity checking multiplies the blind undetected error rate by a factor of $[1-\frac{2(1-p)^{N+1}}{1+(1-2p)^{N+1}}]/[1-(1-p)^N]$; this is plotted for $N=7$ and $0<p<0.5$ at the center of Figure 6.10. For small values of p, one parity bit reduces the undetected error rate by a factor proportional to p, as shown by Taylor's formula:

$$\begin{aligned}\frac{1-\frac{2(1-p)^{N+1}}{1+(1-2p)^{N+1}}}{1-(1-p)^N} &= \frac{1+(1-2p)^{N+1}-2(1-p)^{N+1}}{[1+(1-2p)^{N+1}][1-(1-p)^N]}\\ &= \frac{2\binom{N+1}{2}p^2+O(p^3)}{[2-O(p)][pN+O(p^2)]} \approx \frac{N+1}{2}p,\end{aligned}$$

as $p\to 0$.

Parity checking slows transmission not only because $N+1$ bits need to be transmitted for each N bits of data, but because false alarms occasionally cause good data to be discarded. False alarms increase the volume of rejected strings by a factor of $1+\frac{P_{fa}}{P_{fa}+P_{de}}=1+\frac{2p(1-p)^N}{1-(1-2p)^{N+1}}$. This function is plotted for $N=7$ and $0<p<0.5$ at right in Figure 6.10. As $p\to 0$, Taylor expansions show that the volume increase behaves like

$$\frac{2p(1-p)^N}{1-(1-2p)^{N+1}}=\frac{2p+O(p^2)}{2(N+1)p+O(p^2)}\approx\frac{1}{N+1},$$

This is to be expected, since for tiny p it is very unlikely that more than one bit in a string will be flipped, and the parity bit has one chance in $(N+1)$ of being the one, resulting in a false alarm. However, only the volume of bad-parity data

is increased by false alarms, and since these occur with probability $P_{fa} + P_{de}$, the expected cost of false alarms is an average increase of $P_{fa} \approx p$ bits per string, for $p \approx 0$.

A single parity bit can detect one bit flip in a string of any length N, but this observation is misleading. For any fixed bit-flip probability $p > 0$, the undetected error probability $P_{ue} \to \frac{1}{2}$ as $N \to \infty$ simply because the chance of a double bit flip becomes very great.

Crossed parity checks

Parity bits can locate errors, as well as detect their presence. Suppose that $n \times n$ "data" bits are arranged in a square array of n columns and n rows, and $2n$ parity bits are appended, one to each column and one to each row. The value of each parity bit is chosen to make its row or column sum even. Each data bit is checked by two parity bits, and this cross-checking gives position information when something goes wrong.

Any single bit flip in the data will cause one row and one column parity test to fail, and will indicate the location of the bad data bit. A single flipped parity bit causes an inconsistency that cannot result from a flipped data bit, so it need not cause a false alarm.

If exactly two data bits get flipped, then either two or four parity bits will be inconsistent, but it will not be possible to determine which data bits were bad. Thus, this scheme gives one bit error correction and two bit error detection. What is the probability of an undetected error? For this to happen, all parity checks must be consistent despite some flipped data bits. There is no way for this to happen unless at least three of the $N = n^2 + 2n$ bits are flipped, and only very special arrangements of three or more flips actually results in an undetected error. Let $U(N,k)$ be the number of k-bit flips among N total bits that results in an undetected error. Then $0 \le U(N,k) \le \binom{N}{k}$, and

$$P_{ue} = \sum_{k=3}^{N} U(N,k) p^k (1-p)^{N-k} \le \sum_{k=3}^{N} \binom{N}{k} p^k (1-p)^{N-k}.$$

For fixed N, $P_{ue} = O(p^3)$ as $p \to 0$. Such two-dimensional parity checking costs $2n$ parity bits for n^2 data bits, or $2/n$.

An obvious generalization is to arrange the data into an $n \times n \times n$ cube with $3n^2$ parity bits along three of the faces. To get an undetected error in the three-dimensional setup, at least four bits must be flipped, so $P_{ue} = O(p^4)$ as $p \to 0$. In many cases, though, more data bit flips can be detected and even corrected. The cost is $3/n$ parity bits per data bit. The parity information should be more reliable than the data, if it will be used for correction: the parity bits may be protected with $6n$ parity bits of their own, along the data cube's edges.

Generalizing further, n^d data bits may be arranged in a d-dimensional array with dn^{d-1} parity bits arranged along the coordinate hyperplanes. This will always detect d bit flips, so $P_{ue} = O(p^{d+1})$, at a cost of d/n parity bits per data bit.

Thus, for linear growth in the amount of parity overhead, we obtain geometric decreases in undetected error probability. In addition, for large d, the many-bit-flip combinations giving undetected errors become exceedingly rare.

6.2.2 Hamming codes

Parity checking and cross-checking work by using multiple parity bits to check overlapping subsets of data bits. R. W. Hamming invented a code which identifies the location of a single bad bit by giving its index in binary. Parity bits are sprinkled throughout the codeword and are themselves included in the parity checking. To see how this works, consider an example in which four data bits are protected with three parity bits.

The data bits are in locations 3, 5, 6, and 7, counting from the rightmost bit, which is in location 1. The parity bits are at locations 1, 2, and 4. The parity bit at 1 is chosen so that bits 1, 3, 5, and 7 add up to 0 modulo 2. Parity bit 2 is chosen so that bits 2, 3, 6, and 7 add up to 0 modulo 2, and parity bit 4 is chosen so that bits 4, 5, 6, and 7 add up to 0 modulo 2. Formally, parity bit 2^p is chosen so that it makes all those bits whose location index contains 1×2^p add up to 0 modulo 2, for $p = 0, 1, 2$.

Now suppose that a single bit, data or parity, is flipped in this 7-bit codeword. Let b_0 be the sum of bits 1, 3, 5, and 7 modulo 2, b_1 is the sum of bits 2, 3, 6, and 7 modulo 2, and b_2 is the sum of bits 4, 5, 6, and 7 modulo 2. Then the flipped bit is at position $I = b_0 2^0 + b_1 2^1 + b_2 2^2$, counting from the rightmost bit.

Assuming that at most one bit is flipped, the above provides an error correction method. If $I = 0$, then all the parities are OK and there is no error, otherwise bit I is wrong and should be flipped back to its correct value. Unfortunately, the decoding gives a wrong answer if two or more bits are flipped, so the method is prone to undetected errors.

Linear codes

Hamming's code is an example of a *linear code*. The 16 codewords making up all combinations of four data bits are just linear combinations of the four basic codewords given in Table 6.2, and the codewords are constructed using linearity:

$$\mathbf{c}(\mathbf{x} + \mathbf{y}) = \mathbf{c}(\mathbf{x}) + \mathbf{c}(\mathbf{y}),$$

for any four-bit strings $\mathbf{x}, \mathbf{y}$. For example, the codeword for the data vector $1010 = \mathbf{d}_3 + \mathbf{d}_1$ is $\mathbf{c}(\mathbf{d}_3 + \mathbf{d}_1) = 1001011 + 0011001 = 1010010$. Coordinates are added using arithmetic modulo 2, which is implemented as the bitwise exclusive-or operator in the C programming language. It is straightforward to verify that the parity conditions are preserved.

Linear codes are decoded by linear transformations. We note that the four basic codewords 1001011, 0101010, 0011001, and 0000111 are orthogonal with respect to the usual inner product, if we use addition modulo 2 to add things up. Thus, if we let C be the matrix whose rows are the four basic codewords, then applying the

Data vector	Codeword	7	6	5	4*	3	2*	1*
$\mathbf{d}_3 = 1000$	$\mathbf{c}_3 = \mathbf{c}(1000)$	1	0	0	1	0	1	1
$\mathbf{d}_2 = 0100$	$\mathbf{c}_2 = \mathbf{c}(0100)$	0	1	0	1	0	1	0
$\mathbf{d}_1 = 0010$	$\mathbf{c}_1 = \mathbf{c}(0010)$	0	0	1	1	0	0	1
$\mathbf{d}_0 = 0001$	$\mathbf{c}_0 = \mathbf{c}(0001)$	0	0	0	0	1	1	1

Table 6.2: Basis for Hamming's code with four data and three parity(*) bits.

matrix C to a received codeword, written as a column vector, produces the four expansion coefficients of the data vector:

$$C = \begin{pmatrix} \mathbf{c}_2 \\ \mathbf{c}_2 \\ \mathbf{c}_1 \\ \mathbf{c}_0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

However, we must first fix any errors in the received codeword. To do that, we generate the repair vector $\mathbf{b} = (b_2, b_1, b_0)$ by applying the following matrix to the received codeword:

$$B = \begin{pmatrix} \mathbf{b}_2 \\ \mathbf{b}_1 \\ \mathbf{b}_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

If $\mathbf{b} = \mathbf{0}$, then all is well. Otherwise, interpreting nonzero $\mathbf{b}$ as a three-digit binary number gives the position of the single flipped bit. We flip it back and then apply C to decode.

Notice that $\{\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\} \cup \{\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2\}$ forms an orthogonal basis for the space $\{0,1\}^7$ of seven-bit codewords. We find the error by orthogonal projection onto a three-bit subspace, and then after correcting it find the data by orthogonal projection onto a complementary four-bit subspace.

Hamming's algorithm generalizes to longer codewords. Reserving index 0 to mean "no error" leaves $2^P - 1$ total bits that can be protected by P parity bits, leaving $2^P - P - 1$ data bits. Parity bits are placed at locations 2^p, $0 \le p < P$, counting from rightmost bit 1. The parity bit at 2^p is chosen so that the sum of all codeword bits whose locations contain 1×2^p add up to 0 modulo 2.

Hamming's algorithm divides the codeword bits into P subsets of 2^{P-1} bits each. Computing the parity of the subsets gives a P-bit binary number which, if nonzero, locates the single bit flip in the codeword. The overhead is P parity bits for $2^P - P - 1$ data bits; this overhead becomes negligible very fast as $P \to \infty$. Thus, as for one-parity-bit single-error detection, single-error correction costs arbitrarily little for long codewords.

Any single-bit correction code must add at least $\log_2 N$ bits of overhead to N bits of data, since that is the least information needed to locate a flipped bit. Hamming's algorithm is only one of many ways of inserting this overhead, but it gives an elegant and simple decoding algorithm, and simultaneously protects both parity and data bits.

Hamming distance

Can $k > 1$ bit flips be corrected? Hamming used a distance function on bit strings to make this question geometric. For integer $k > 1$, there is a set of codewords that are sufficiently distant from each other so that flipping k bits in any one of them still leaves the correct codeword as the nearest. This *restoration to the nearest codeword* error correction scheme fixes up to k flipped bits.

The Hamming distance function *dist* counts the number of bits at which two bit strings of equal length differ. For example, $\text{dist}(010011101, 010101110) = 4$. Two bit strings are identical if and only if the distance between them is zero.

For each string c of N bits, we may count the number of codewords at each Hamming distance k from c. There are $\binom{N}{k}$ of them, corresponding to the number of ways we may choose the k bits that differ. The *Hamming sphere* of radius r centered at a bit string c of length N is defined to be the set of bit strings d of length N satisfying $\text{dist}(c, d) \leq r$. The *volume* enclosed by the Hamming sphere is the total number of bit strings within it, and is found by adding up the counts for distances $0, 1, \ldots, r$:

$$\text{Vol}_N(r) \stackrel{\text{def}}{=} \sum_{k=0}^{r} \binom{N}{k}. \tag{6.8}$$

Thus $\text{Vol}_N(0) = 1$ for every $N \geq 1$, and $\text{Vol}_N(r) = 2^N$ for every $r \geq N$.

We may visualize bit strings as the vertices of an N-dimensional unit hypercube. The vertex at the origin corresponds to $00\cdots0$, the string of N zero bits, and the vertex farthest from the origin, in both the Hamming and the Euclidean distance, corresponds to the string $11\cdots1$ of N ones. Edges connect adjacent vertices, such as $00\ldots0$ and $10\ldots0$ on the x-axis, if and only if their Hamming distance is 1. Since each vertex in this N-cube has edges to N other vertices, we may estimate

$$\text{Vol}_N(r) \leq 1 + N + \cdots + N^r < N^{r+1}. \tag{6.9}$$

This is very crude, but it is accurate enough for our purposes.[5]

The distance between a codeword c and a set C' of equal-length bit strings is $\text{dist}(c, C') \stackrel{\text{def}}{=} \min\{\text{dist}(c, c') : c' \in C'\}$. Thus $c \in C'$ if and only if $\text{dist}(c, C') = 0$. Likewise, the distance between two sets C and C' of equal-length bit strings is $\text{dist}(C, C') \stackrel{\text{def}}{=} \min\{\text{dist}(c, c') : c \in C, c' \in C'\}$, with $C \cap C' \neq \emptyset$ if and only if $\text{dist}(C, C') = 0$. Now suppose we take the set of N-bit strings and select from it a collection of disjoint Hamming spheres of radius r. If we use the bit strings in the centers of those spheres as codewords, then we will be able to correct up to r bit flips by restoration to the nearest codeword.

The simplest example of this idea is the *repetition* code. Suppose a positive integer r is given. We may encode the two-letter alphabet $\{0, 1\}$ by $2r+1$ repetitions:

$$\mathbf{c}(0) = \overbrace{00\cdots0}^{2r+1}; \qquad \mathbf{c}(1) = \overbrace{11\cdots1}^{2r+1}.$$

[5]The solution to Exercise 3 of Chapter 4 provides the sharp estimate $\text{Vol}_N(r) = O(N^r)$.

These two codewords are diagonally opposite corners of a $(2r+1)$-dimensional unit hypercube, with $\mathbf{c}(0)$ at the origin. Then $\text{dist}(\mathbf{c}(0), \mathbf{c}(1)) = 2r+1$, so we can correct r or fewer flipped bits by accepting the majority. This is a linear code, and the decoding algorithm consists of applying the matrix $C = (1 \quad 1 \quad \cdots \quad 1)$, using ordinary real arithmetic. If the single scalar output is less than r, we record 0 as the received bit. If the output is $r+1$ or greater, we record 1.

The repetition code is terribly inefficient, however. It costs $2r$ parity bits to protect just one data bit, although they protect themselves at the same time. The inefficiency stems from the small number of codewords. If there are $M = 2^m$ codewords in the code, then we may transmit $m = \log_2 M$ data bits per codeword. If the codewords are N bits long, then we may say that there are $p = N - m$ parity bits included in each. This is a reasonable cost assessment even if we choose not to identify particular bits as parity bits or data bits. One way to increase efficiency is therefore to increase the number of codewords that are protected against r or fewer flipped bits. We must control the growth in codeword bit length N as we build larger codes, so that $p = N - m$, or at least p/N, tends to some finite limit.

In fact, there are codes for which $p/N \to 0$ as $N \to \infty$. With our hypercube intuition, we may describe their construction as follows:

Theorem 6.12 (E. N. Gilbert, 1952) *For any positive integer r and any $\epsilon > 0$, there is an integer N and a binary code consisting of 2^m N-bit codewords such that*

1. *restoration to the nearest codeword corrects up to r bit flips;*
2. *we may transmit $m > (1-\epsilon)N$ data bits per codeword, with fewer than ϵN of the bits being needed for error correction.*

Proof: Suppose we choose codewords from the set of all N-bit strings, which we imagine as the vertices of the unit N-cube. We begin with the origin, and remove the $\text{Vol}_N(2r)$ vertices within Hamming distance $2r$ of it. For the second and subsequent codewords, we choose one of the remaining vertices and remove its Hamming sphere of $\text{Vol}_N(2r)$ vertices. We may do this at least $M = \lfloor 2^N / \text{Vol}_N(2r) \rfloor$ times, since there were 2^N bit strings to begin with. The M codeword bit strings thus chosen will be our code. It evidently satisfies condition 1, as each codeword is at least Hamming distance $2r+1$ away from all the others.

We now count M using Inequality 6.9:

$$M = 2^m \geq \left\lfloor \frac{2^N}{N^{2r+1}} \right\rfloor > \frac{2^N}{N^{2r+2}}, \quad \Rightarrow m > N\left[1 - (2r+2)\frac{\log_2 N}{N}\right].$$

Since $(\log_2 N)/N \to 0$ as $N \to \infty$, we may choose N large enough so that $m \geq (1-\epsilon)N$. □

Note that some bit strings might be outside the Hamming spheres of the codewords generated by Theorem 6.12. If the receiver gets one of these, which cannot be decoded, it may declare a detected error and request retransmission.

Gilbert's coding theorem is constructive, but it does not produce a linear code in general. It also does not provide a decoding algorithm other than a table search through a possibly large number of codeword strings.

Although the fraction of parity bits needed to correct up to r bit flips can be made arbitrarily small, the price is using long codewords which are likelier to have multiple bit errors. There is also a lower bound on the absolute number of parity bits needed to correct up to r bit flips, generalizing the 1-bit correction bound mentioned earlier:

Theorem 6.13 *Any N-bit code that can correct r bit flips by restoration to the nearest codeword must consume at least $p = \log_2 \mathrm{Vol}_N(r)$ bits of each codeword for parity information.*

Proof: Let m be the number of data bits, so $M = 2^m$ is the number of codewords and $p = N - m$ is the number of parity bits. Each codeword is the center of a Hamming sphere containing $\mathrm{Vol}_N(r)$ vertices. These spheres are disjoint, and the union of all of them makes up at most 2^N vertices. Thus $M\mathrm{Vol}_N(r) \leq 2^N$, and so $p = N - m \geq \log_2 \mathrm{Vol}_N(r)$. □

Note that $\log_2 \mathrm{Vol}_N(r) \leq (r+1)\log_2 N = O(\log N)$ by Inequality 6.9.

6.2.3 Checksums and cyclic redundancy codes

An old method for checking arithmetic, which was once taught in elementary schools, is *casting out nines*. We observe that if $d_n \cdots d_1 d_0$ is the base-ten representation of the integer $x > 0$, where $0 \leq d_k < 10$ for all $k = 0, 1, \ldots, n$ and

$$x = d_n \times 10^n + \cdots + d_1 \times 10 + d_0, \tag{6.10}$$

then

$$x = d_n + \cdots + d_0 \pmod 9. \tag{6.11}$$

In other words, the sum of the decimal digits of x has the same remainder as x after division by 9. This is proved using the congruence $10 = 1 \pmod 9$, which implies $10^k = 1 \pmod 9$ for every integer $k \geq 0$, to reduce Equation 6.10. We may write $x' = d_n + \cdots + d_0$ and expand $x' = d'_{n'} \times 10^{n'} + \cdots + d'_0$ and apply Equation 6.11 again to get $x = x' = x'' = \cdots \pmod 9$. We continue until we have a single digit, for then the values stop changing. Let $c = c_9(x)$ be this digit. For example, $c_9(1\,872) = 9$ and $c_9(22\,883) = 5$.

This digit is called the *checksum*. In the C programming language, its formula would be $c_9(x) = x\%9$, although the limited integer sizes in most common computers limit the domain to numbers x with ten or fewer decimal digits. However, the special properties of the modulus 9 allow us to compute $c_9(x)$ for much larger x presented as a string of decimal digits, using the following algorithm:

Casting-Out-Nines Checksum

```
castout9( digit[], L ):
[0] Initialize chksum = 0
[1] For k=0 to L-1, replace chksum = (chksum + digit[k])%9
[2] Return chksum
```

The checksum from casting out nines can be used to detect addition errors, since $c_9(x+y) = c_9(x) + c_9(y) \pmod 9$, so $c_9(x+y+\cdots) = c_9(c_9(x) + c_9(y) + \cdots)$. It can also check products, since $c_9(xy\cdots) = c_9(c_9(x)c_9(y)\cdots)$. For example, the product $1\,872 \times 22\,883$ is not equal to 42 846 976, since $c_9(42\,846\,976) = 1$ whereas $c_9(c_9(1\,872)c_9(22\,883)) = c_9(9 \times 5) = 9$. However, the method does not locate the error. Also, it fails to detect some errors: the product $1\,872 \times 22\,883$ is not equal to 42 846 966 either, even though $c_9(42\,846\,966) = 9$. If our numbers are random in the sense that $c_9(x)$ is uniformly distributed over $\{1, 2, \ldots, 9\}$, then casting out nines has an 8/9 probability of catching an error. Still, one such error in nine goes undetected.

We could also use "casting out elevens" to compute a checksum $c_{11} = c_{11}(x)$ satisfying $x = c_{11}(x) \pmod{11}$. But $10 = -1 \pmod{11}$, so

$$10^k = \begin{cases} 1 \pmod{11}, & \text{if } k \text{ is even,} \\ -1 \pmod{11}, & \text{if } k \text{ is odd.} \end{cases}$$

Thus to compute $c_{11}(x)$, we must add the even-position digits starting with digit 0, and then subtract the odd-position digits. For example, $c_{11}(1\,872) = (8+2) - (1+7) = 2$ and $c_{11}(22\,883) = (2+8+3) - (2+8) = 3$. Note that $c_{11}(-1\,872) = -2$, as all digits keep the sign of x. The process is repeated until the result lies in the range $[-10, 10]$. If it is not positive, 11 is added to get a result in the range $[1, 11]$. This method may be applied to large numbers x presented as long arrays of decimal digits:

Casting-Out-Elevens Checksum

```
castout11( digit[], L ):
[0] Initialize chksum = 0
[1] For k=0 to L-1, do [2] to [3]
[2]    If k is even, let chksum = (chksum+digit[k])%11
[3]    Else let chksum = (chksum + 11 - digit[k])%11
[4] Return chksum
```

We add 11 in step 3 to guarantee that the left-hand operand of % is never negative. It is an easy exercise to prove that c_{11} detects all one-digit errors in decimal arithmetic. However, one multiple-digit error in 11 will go undetected using this checksum. Notice that c_{11} catches the error $1\,872 \times 22\,883 \neq 42\,846\,966$, since $c_{11}(42\,846\,966) = (2+4+9+6) - (4+8+6+6) = -3 = 8 \pmod{11}$, and $8 \neq 2 \times 3$. In general, we can combine checksums to decrease the probability of undetected errors.

Checksums are also used to detect corrupted data. Both x and $y = c(x)$ are stored, and then c is recomputed from the retrieved x and compared to the retrieved y. If they disagree, then either x or y was corrupted during storage. If they agree, there is a high probability that neither was corrupted.

Humans use casting out nines because one-digit decimal arithmetic is considered easy, so the effort of checking is low. Digital computers can perform multi-bit binary arithmetic easily, and analogs of casting out nines exist to check calculations and data. We expect different kinds of errors than humans make. For example, a single bit might be flipped, or a segment of bits might be set all to zero, due to faulty memory or a bad connection. So, if x is the nonnegative integer representing a string of bits, we may wish to compute a checksum $c(x)$ with the following desirable properties:

Checksum Goals

CSG-1: Any single-bit error is detected.

CSG-2: Any double-bit error is detected.

CSG-3: Certain special kinds of errors like truncations of the bit string are detected with probability $1 - \epsilon$.

CSG-4: Any multiple-bit error is detected with probability $1 - \epsilon$.

The undetected error probability, ϵ, will depend on the formula used for $c(x)$. With the casting out nines checksum, $\epsilon = 1/9 \approx 11\%$. It is dependent on the range of values taken by $c(x)$, which is a measure of the amount of information that $c(x)$ retains from x and so can be measured in bits. If $c(x)$ takes all values in $\{0, 1, \ldots, 2^{16} - 1\}$, we will call it a 16-bit checksum, and so on.

Let $X = X(s) = c^{-1}(s)$ be the set of bit strings with checksum s. If properties 1 and 2 hold, then we must have it that the Hamming distance $\text{dist}(X(s), X(s')) \geq 2$ whenever $s \neq s'$. Such a code contains enough information to correct one flipped bit, so it must contain $\log_2 N$ bits of redundancy for N bits of data. Hence, a 16-bit checksum can protect at most 2^{16} data bits, which is 8 kilobytes, and a 32-bit checksum can protect at most 512 megabytes.

To accomplish CSG-3, detecting truncations, the number of data bits can be transmitted as a second checksum, or better yet appended to the data bits before the checksum or CRC is computed. The receiver compares the portion of the received bits which contains the length to the actual number received. Any inconsistency is cause for rejection. In the 16-bit case, with the length appended to the data, the probability is only $1/2^{16} \approx .000015$ that chopping off the end exposes bits claiming the right length. Thus truncation is detected with probability $1 - \epsilon \approx 99.9985\%$, assuming that all 16-bit strings are equally probable in any 16-bit segment of the data.

CSG-4 can be restated as follows: the probability that $c(x) = c(x')$ given that $x \neq x'$ must be less than ϵ. Using values of s to partition the space of bit strings, this requires that $c^{-1}(s)$ contains fewer than ϵK of the K total bit strings. This implies that the number of checksum values — call it S — must be larger than

$1/\epsilon$, and that ϵ must be larger than $1/S$. 16-bit checksums thus at best allow $1 - \epsilon \approx 99.998\%$.

Note that achieving CSG-4 does not necessarily achieve CSG-3, since truncations are a tiny subset of the possible errors and may, in principle, have a high probability of going undetected without defeating goal 4.

Modular implementations

Let $0 \leq x < 2^N$ be the number represented by an N-bit string. For each fixed integer $S > 1$, consider the *modular checksum* $c_S(x) = x\%S$ given by the remainder after division by S. If b is a positive integer such that $2^{b-1} < S \leq 2^b$, then $0 \leq c_S(x) < 2^b$ and we will say that c_S gives a b-bit checksum. In any case, $c_S(x) = x \pmod{S}$, so two bit strings $x \neq y$ are distinguishable from just this checksum if and only if $x \neq y \pmod{S}$. Such remainder checksums are easy to analyze because $c_S(x+y) = c_S(x) + c_S(y)$ and $c_S(xy) = c_S(x)c_S(y)$ for all integers x, y. We now find some values of S that achieve checksum goals 1–4.

To meet CSG-1, we must have that $x - y$ is not a multiple of S if x and y differ at exactly one bit. But in that case, $x - y = \pm 2^k$ for some integer $0 \leq k < N$, so by unique factorization, $S|(x-y)$ only if S is an integer power of 2. Hence any $S > 1$ which is not an integer power of 2 will satisfy CSG-1. In particular, $S = 2^b - 1$ gives a b-bit checksum for every $b > 0$, and since $2^b = 1 \pmod{S}$, we can simply add the b-bit "digits" of x to compute $c_S(x)$, just as in casting out nines. Similarly, if $S = 2^b + 1$, then $2^b = -1 \pmod{S}$, and we can use an alternating sum as in casting out elevens.

For $b = 8$ on a typical computer that uses 8-bit characters and 32-bit integers, we may us the following functions to compute the $S = 255$ and $S = 257$ modular checksums:

Checksum Modulo 255

```
castout255( octet[], L ):
[0] Initialize chksum = 0
[1] For k=0 to L-1, do [2]
[2]    Replace chksum = (chksum+octet[k])%255
[3] Return chksum
```

Checksum Modulo 257

```
castout257( octet[], L ):
[0] Initialize sE = 0 and sO = 0
[1] For k=0 to L-1, do [2] to [3]
[2]    If k is even, then replace sE = (sE+octet[k])%257
[3]    Else k is odd, so replace sO = (sO+octet[k])%257
[4] Let chksum = sE - sO
[5] If chksum<0, then increment chksum += 257
[6] Return chksum
```

To meet CSG-2, note that if the N-bit integers x and y differ at exactly two bits, then $x - y = \pm 2^j(2^k \pm 1)$, for some integers $0 \leq j < N$ and $0 < k < N$. If $c_S(x) = c_S(y)$, then S divides $2^j(2^k \pm 1)$. Hence, to achieve CSG-2, it suffices to find an S which is not an integer power of 2 and does not divide $2^k \pm 1$ for any $0 < k < N$. There are just $2N$ such numbers, so it is a simple matter of performing $O(N)$ trial divisions[6] to find an S that avoids dividing any of them. We are sure to find a b-bit non-divisor if $2^b > N$. Unfortunately, this excludes the fast implementation choices $S = 2^b \pm 1$. In the next section, we will modify the arithmetic to obtain a fast algorithm despite this prohibition.

Checksum goals 3 and 4 are accomplished by choosing $S > 1/\epsilon$. If all N-bit string differences $x - y$ are equally probable, then the chances that $x \neq y$ but $c_S(x-y) = 0$ are just $1/S < \epsilon$, and no particular multi-bit error is more likely than any other.

Mod-2 polynomials — arithmetic without carries

The special values $S = 2^b \pm 1$ for a modular checksum c_S give fast algorithms: each block of b bits contributes its share to the checksum independently of the other blocks. For computers with fixed integer bitwidth b, this means no carries are needed. However, having just two choices is a big limitation, especially as both choices interfere with CSG-2. This problem can be overcome with alternative arithmetic. Instead of treating bit strings as ordinary numbers written in binary, we consider them to be coefficients of a polynomial and manipulate them using the rules for polynomial algebra.

Polynomials whose coefficients are just 0 or 1 can be added and multiplied termwise, using arithmetic modulo 2. We shall call these objects *mod-2 polynomials* and use the qualifier *mod-2* to designate the new arithmetic operations. For example, if $p(t) = t^2 + t + 1$ and $q(t) = t + 1$ are mod-2 polynomials, then their mod-2 sum is $p(t) + q(t) = t^2$, since $1 + 1 = 2 = 0 \pmod 2$ and $t + t = 2t = 0t = 0 \pmod 2$. Likewise, their mod-2 product is $p(t)q(t) = t^3 + 1$, which we get by adding $t^3 + t^2 + t$ to $t^2 + t + 1$. The coefficients of these mod-2 polynomials determine numbers in binary: p is 111 (base 2) = 7 (base 10), and q is 11 (base 2) = 3 (base 10), so these polynomial operations give an alternative addition and multiplication in which $7 + 3 = 4$ and $7 \times 3 = 9$. The mod-2 values can also be determined by reconsidering the coefficients as integers and evaluating the polynomial at 2: $p(2) = 5$ and $q(2) = 3$. Likewise, given a nonnegative integer, we can find its mod-2 polynomial by computing its binary expansion. Table 6.3 shows the addition and multiplication facts for the first eight of these mod-2 polynomials.

Mod-2 polynomial addition may also be interpreted as componentwise addition in the vector space of sequences whose components are the integers modulo 2. Addition in this space is the same as ordinary binary addition, only without carries. Hence, addition may be performed on each bit independently, using the addition facts $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1$, and $1 + 1 = 0$. Such carry-free addition is the

[6]But half of the numbers $\{2^k \pm 1 : 1 \leq k \leq N\}$ will have at least $N/2$ digits, so the cost of testing S is actually $O(N^2)$.

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	8	10	12	14
3	0	3	6	5	12	15	10	9
4	0	4	8	12	16	20	24	28
5	0	5	10	15	20	17	30	27
6	0	6	12	10	24	30	20	18
7	0	7	14	9	28	27	18	21

Table 6.3: Arithmetic with the first eight mod-2 polynomials. Left: Addition table. Right: Multiplication table.

same as the bitwise exclusive-or operation, which in the C programming language is denoted by the caret (^) operator. In the following implementation, a new array is allocated to hold the sum, and the exclusive-or assignment operator ^= is used to accumulate the mod-2 sums in it:

Mod-2 Polynomial Sums

```
mod2polysum( p[], dp, q[], dq ):
[0] Let ds = max(dp,dq) and allocate sum[0],...,sum[ds]
[1] For d=0 to ds, let sum[d] = 0
[2] For d=0 to dp, replace sum[d] ^= p[d]
[3] For d=0 to dq, replace sum[d] ^= q[d]
[4] Return sum[] and ds
```

Putting the operands into arrays allows arbitrary degree polynomials, but is inefficient. However, if the degrees dp and dq of the summand polynomials are both less than the bitwidth of an unsigned integer type known to the computer, then they may be encoded as integers by evaluating $p(2)$ by Horner's method:[7]

Convert an Integer-Size Mod-2 Polynomial to an Integer

```
mod2polyint( p[], dp ):
[0] Initialize ip = 0
[1] For d=dp down to 0, let ip = 2*ip + p[d]
[2] Return ip
```

The sum of mod-2 polynomials that fit into integers may be performed by a single bitwise exclusive-or operation:

Integer-Size Mod-2 Polynomial Sums

```
intmod2polysum( p, q ):
[0] sum = p^q
[1] Return sum
```

[7] A more robust implementation would test the size of dp, or else there could be a silent integer overflow and wrong return value.

Mod-2 polynomial multiplication is given by combining index shifts with ands and exclusive-ors. Using arrays of bits, this may be implemented as follows:

Mod-2 Polynomial Products

```
mod2polyproduct( p[], dp, q[], dq ):
[0] Let ds = dp+dq and allocate prod[0],...,prod[ds]
[1] For d=0 to ds, let prod[d] = 0
[2] For i=0 to dp, do [3]
[3]    For j=0 to dq, replace prod[i+j] ^= p[i]&q[j]
[4] Return prod[] and ds
```

If $ds = dp + dq$ is small enough, then all of the mod-2 polynomials will fit into some integer type and the above may be performed more efficiently with bit shifts and bitwise exclusive-ors. The C programming language supplies the operators `<<=` and `>>=` for bitshift-left and bitshift-right assignment, respectively:

Integer-Size Mod-2 Polynomial Products

```
intmod2polyproduct( p, q ):
[0] Initialize prod = 0
[1] While p>0, do [2] to [4]
[2]    If p is odd, then replace prod ^= q
[3]    Replace q <<= 1 by its bitshift left
[4]    Replace p >>= 1 by its bitshift right
[5] Return prod
```

The *degree*, written $\deg p$, of a mod-2 polynomial $p = p(t)$ is defined as for ordinary polynomials: it is the highest power of t present in $p(t)$. The degree of the zero polynomial is undefined, though it can be set to $-\infty$ with the convention that $(-\infty) + d = d + (-\infty) = -\infty$ for any integer d. The degree function then satisfies two equations:

$$\begin{aligned} \deg(p+q) &\leq \max\{\deg p, \deg q\}; \\ \deg(pq) &= \deg p + \deg q. \end{aligned}$$

In practice, a degree function may return -1 as the degree of the zero polynomial:

Mod-2 Polynomial Degree

```
mod2polydegree( p[], dp ):
[0] For d=dp down to 0, do [1]
[1]    If p[d]==1, then return d
[2] Return -1
```

An initial estimate dp of the degree must be supplied. If this is small enough, then the mod-2 polynomial will fit into an integer and we can be more efficient:

Integer-Size Mod-2 Polynomial Degree

```
intmod2polydegree( p ):
[0] Initialize degree = -1
[1] While p>0, do [2] to [3]
[2]    Bitshift right p >>= 1
[3]    Increment degree += 1
[4] Return degree
```

The division theorem for ordinary polynomials applies to mod-2 polynomials:

Theorem 6.14 *For any mod-2 polynomials p_1 and $p_2 \neq 0$, there are unique mod-2 polynomials q, r such that $p_1(t) = p_2(t)q(t) + r(t)$ and $0 \leq \deg r < \deg p_2$.*

Proof: For existence, we construct the quotient q and remainder r using synthetic division. The following implementation produces arrays of 0s and 1s representing the mod-2 polynomials q, r, given input arrays `p1[0]`,...,`p1[d1]` and `p2[0]`,...,`p2[dd]`:[8]

Mod-2 Polynomial Quotient and Remainder

```
mod2polydivision( p1[], d1, p2[], d2 ):
[0] If d1<d2, then let dq = -1 and let dr = d1
[1] Else do [2] to [5]
[2]    Let dq = d1-d2 and let dr = d2-1
[3]    For d=d1 down to d2, do [4] to [5]
[4]       If p1[d]==1, then do [5]
[5]          For n=0 to d2, replace p1[n+d-d2] ^= p2[n]
[6] Let dr = mod2polydegree(p1[], dr)
[7] Return p1[], dq, dr
```

Upon termination, the returned array p_1 will contain the coefficients of the remainder and quotient mod-2 polynomials as subarrays: `p1[0]`, ..., `p1[dr]` will be r, while `p1[d2]`, ..., `p1[d2+dq]` will contain q. If the returned value $dr = -1$, then the remainder r is zero and its part of the array is not present. Likewise, if $dq = -1$, then the quotient q is zero and it is not present in the returned array p_1.

For uniqueness, suppose that $p_1(t) = p_2(t)q_1(t) + r_1(t) = p_2(t)q_2(t) + r_2(t)$ with $0 \leq \deg r_1 < \deg p_2$ and $0 \leq \deg r_2 < \deg p_2$. Then $p_2(t)[q_2(t) - q_1(t)] = r_1(t) - r_2(t)$. Suppose toward contradiction that $q_1 \neq q_2$. Then $p_2[q_2 - q_1] \neq 0$, so we may compute mod-2 degrees: $\deg(r_1 - r_2) = \deg(p_2[q_2 - q_1]) = \deg p_2 + \deg(q_2 - q_1) \geq \deg p_2$, but this contradicts $\deg(r_1 - r_2) \leq \max\{\deg r_1, \deg r_2\} < \deg p_2$. Hence we must have $q_1 = q_2$, so $r_1 - r_2 = p_2[q_2 - q_1] = 0$, so $r_1 = r_2$. □

We will reuse symbols and denote the quotient and remainder by $p(t)/q(t)$ and $p(t) \,\%\, q(t)$.

[8] This function assumes that the denominator polynomial p_2 is nonzero, or else the results will be nonsense. How could this assumption be tested in practice?

/, %	1	2	3	4	5	6	7	8
1	1,0	0,1	0,1	0,1	0,1	0,1	0,1	0,1
2	2,0	1,0	1,1	0,2	0,2	0,2	0,2	0,2
3	3,0	1,1	1,0	0,3	0,3	0,3	0,3	0,3
4	4,0	2,0	3,1	1,0	1,1	1,2	1,3	0,4
5	5,0	2,1	3,0	1,1	1,0	1,3	1,2	0,5
6	6,0	3,0	2,0	1,2	1,3	1,0	1,1	0,6
7	7,0	3,1	2,1	1,3	1,2	1,1	1,0	0,7
8	8,0	4,0	7,1	2,0	2,2	3,2	3,1	1,0

Table 6.4: Quotients q and remainders r from division $y(t) = q(t)x(t) + r(t)$ of mod-2 polynomials x, y expressed as base 10 versions of their coefficient bit strings. Here x is at the top of the column and y is at the left of the row.

If both the numerator and denominator polynomials fit into an integer type, then there is a more efficient mod-2 division implementation using bit-shifts:

Integer-Size Mod-2 Polynomial Quotient and Remainder

```
intmod2polydivision( p1, p2 ):
[0] Initialize q=0 and  r=p1
[1] Let sh = intmod2polydegree(r)-intmod2polydegree(p2)
[2] If sh >= 0, then do [3] to [4]
[3]    Replace r ^= (p2<<sh) and replace q ^= (1<<sh)
[4]    Go to [1]
[5] Return q, r
```

Table 6.4 shows the quotients and remainders for the first eight nonzero mod-2 polynomials.

If the remainder is zero, so that $p_1(t) = p_2(t)q(t)$, then we say that p_2 *divides* p_1. Having Theorem 6.14, we can implement Euclid's algorithm for mod-2 polynomials, so each pair $x = x(t), y = y(t)$, not both zero, will have a unique greatest common divisor not equal to zero. In the following implementation, derived from the one on page 4, we assume without loss that x is not the zero polynomial:

Euclid's Algorithm for Mod-2 Polynomials

```
mod2polygcd( x[], dx, y[], dy ):
[0-] Let dz = dx and allocate z[0],...,z[dz]
[0]  For d=0 to dx, copy z[d] = x[d]
[1]  Compute dr,dq,y[] with mod2polydivision(y[],dy,x[],dx)
[1+] Let dx = dr, and for d=0 to dr, copy x[d] = y[d]
[2]  Let dy = dz, and for d=0 to dz, copy y[d] = z[d]
[3]  If dr>0, then go to [0]
[4]  Return y[], dy
```

The more efficient implementation for integer-size mod-2 polynomials is left as an exercise. We may reuse the notation gcd() for the result.

Now $\gcd(x, y)$ is again a mod-2 polynomial. We will say that x, y are *relatively prime* if $\deg \gcd(x, y) = 0$. We will say that a mod-2 polynomial x of positive degree is *irreducible* if x is relatively prime to every mod-2 polynomial y of strictly smaller degree.[9] Equivalently, x is irreducible if no mod-2 polynomial of smaller positive degree divides x.

It is straightforward to prove the following generalizations to mod-2 polynomials of Lemmas 1.1 and 1.3:

Lemma 6.15 *If* $\deg y \geq 0$ *and* $x(t)$ *divides* $y(t)$, *then* $\deg x \leq \deg y$.

Proof: Write $y(t) = x(t)q(t)$ and note that $q \neq 0$ since $\deg y \geq 0 \Rightarrow y \neq 0$. But then $\deg y = \deg qx = \deg q + \deg x$, so $\deg y - \deg x = \deg q \geq 0$, so $\deg y \geq \deg x$. □

Lemma 6.16 *If* $z(t)$ *divides* $x(t)y(t)$ *and* x, z *are relatively prime mod-2 polynomials, then* $z(t)$ *divides* $y(t)$. □

The proof is left as an exercise.

Theorem 6.17 *Every mod-2 polynomial* p *can be factored into irreducible mod-2 polynomials, and the factorization is unique except for the order. Namely, if* $p(t) = p_1(t) \cdots p_n(t) = q_1(t) \cdots q_m(t)$ *are two factorizations, then* $n = m$ *and, possibly after re-indexing,* $p_i = q_i$ *for all* $i = 1, \ldots, n$. □

The proof is substantially the same as that for Theorem 1.4. It leads immediately to the following:

Corollary 6.18 *If an irreducible mod-2 polynomial* $z(t)$ *divides a product of mod-2 polynomials* $x(t)y(t)$, *then either* $z(t)$ *divides* $x(t)$ *or* $z(t)$ *divides* $y(t)$. □

To check whether a mod-2 polynomial $x(t)$ of degree $d = \deg(x) > 0$ is irreducible, it is only necessary to perform trial divisions with irreducible mod-2 polynomials of degree 1 through $\lceil d/2 \rceil$. There are fewer than $2^{d/2}$ of those, and except for $t \leftrightarrow 2$ they all correspond to odd numbers. It is easy to list all these trial divisors if the candidate irreducible mod-2 polynomial is integer-sized:

Test if an Integer-Size Mod-2 Polynomial is Irreducible

```
intmod2polyirreducible( x ):
[0] Let d2 = intmod2polydegree(x)
[1] For y=2 and odd y=3 to (1<<d2)-1, do [2] to [3]
[2]    Compute remainder r with intmod2polydivision(x,y)
[3]    If r==0, then return FALSE
[4] Return TRUE
```

[9] An irreducible polynomial is like a prime number, and would be called a prime if it were not for the possible confusion with ordinary primes. For example, the irreducible mod-2 polynomial $S(t)$ used in CRC-32 corresponds to the composite number $S(2) = 4374732215 = (5)(7)(29)(4310081)$.

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	5	4	7	2	1
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	1	4	3	2	5

Table 6.5: Left: Arithmetic with the eight mod-2 polynomials, reduced modulo t^3. Left: Addition table, same as before. Right: Multiplication table.

The returned value will be TRUE if the mod-2 polynomial is irreducible, or FALSE if a factor of degree 1 or more is found.

Mod-2 polynomials modulo a mod-2 polynomial

Yet another kind of modular arithmetic is possible, using the set of mod-2 polynomials reduced modulo a fixed mod-2 polynomial $S = S(t)$. For example, Table 6.5 shows the entries of Table 6.3, reduced modulo the polynomial $S(t) = t^3$. In this kind of arithmetic there are only finitely many distinct elements, in fact $2^{\deg S}$ of them, with $\deg S = 3$ in the example. That is because the degree of the remainder mod-2 polynomial after division by S must be strictly less than $\deg S$, which means it can have at most $\deg S$ coefficients, and there are exactly $2^{\deg S}$ such mod-2 polynomial remainders. Addition and multiplication are commutative, associative, and distributive just like for ordinary integers. Each element has a unique additive inverse, since there is a unique 0 in each row and column. However, if S is reducible then its factors will not have multiplicative inverses. In the example, $S(t) = t^3$ is evidently reducible, and the factor t^2 corresponding to 4 has no multiplicative inverse. This is the only source of trouble: an irreducible polynomial such as $S(t) = t^3 + t + 1$ will give every nonzero element a unique multiplicative inverse as well. Table 6.6 illustrates how: there is a unique 1 in each nonzero row and column of the multiplication table.

Now suppose that $S = S(t)$ is a fixed mod-2 polynomial. We define the checksum $c_S(x) = x(t) \,\%\, S(t)$ to be the remainder polynomial, of degree less than $b = \deg S$, produced by the `polydiv2()` function. The coefficients of that remainder will be a b-bit checksum for x. For example, Table 6.7 shows the checksums of the 32 five-bit strings, namely the 32 mod-2 polynomials of degree four or less, reduced modulo the polynomial $S(t) = t^3 + t + 1$.

Note that mod-2 polynomial checksums may be computed very efficiently with just bit-shift and exclusive-or operations. It is not necessary to keep the quotient polynomial, and the degree of the modulus mod-2 polynomial $S(t)$ limits the number of bits that must be stored. If there are b such bits, for $\deg S = b$, then the

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

Table 6.6: Arithmetic with the first eight mod-2 polynomials, reduced modulo $t^3 + t + 1$. Left: Addition table, same as always. Right: Multiplication table.

x	x base 2	$c_S(x)$	x	x base 2	$c_S(x)$
0	00000	0	16	10000	6
1	00001	1	17	10001	7
2	00010	2	18	10010	4
3	00011	3	19	10011	5
4	00100	4	20	10100	2
5	00101	5	21	10101	3
6	00110	6	22	10110	0
7	00111	7	23	10111	1
8	01000	3	24	11000	5
9	01001	2	25	11001	4
10	01010	1	26	11010	7
11	01011	0	27	11011	6
12	01100	7	28	11100	1
13	01101	6	29	11101	0
14	01110	5	30	11110	3
15	01111	4	31	11111	2

Table 6.7: Three-bit checksums for all strings of five bits, reduced modulo $S(t) = t^3 + t + 1$.

total number of operations will be $O(bN)$ for N bits of data. Note that in the following implementation, the bit strings are stored in reverse order, with the first or highest-order bit occupying index 0, and the last or lowest-order bit at index nb:

Compute a Mod-2 Polynomial Checksum

```
mod2polychecksum( msg[], nb, mod2poly[], dm ):
[0] For n=0 to nb-dm-1, do [1] to [2]
[1]    If msg[n]==1, then do [2]
[2]       For d=0 to dm, replace msg[n+d] ^= mod2poly[dm-d]
[3] Let chksum = 0
[4] For n=nb-dm to nb-1, replace chksum = 2*chksum+msg[n]
[5] Return chksum
```

For CSG-1, suppose that $x(t)$ and $y(t)$ are mod-2 polynomials that differ at exactly one coefficient. Then $x(t) - y(t) = t^k$ for some nonnegative integer k, so $c_S(x) = c_S(y)$ if and only if $S(t)$ divides t^k. If $S(t)$ is irreducible and $\deg S > 1$, then this is impossible: $S(t)$ must divide t by Corollary 6.18, but then Lemma 6.15 gives the contradiction $\deg S \leq \deg t = 1$. For such S, c_S will detect all 1-bit errors.

For CSG-2, suppose that $x(t)$ and $y(t)$ differ at exactly two coefficients. Then $x(t) - y(t) = t^j(t^k + 1)$ for some integers $j \geq 0$ and $k > 0$. Thus $c_S(x) = c_S(y)$ if and only if $S(t)$ divides $t^j(t^k + 1)$ as a mod-2 polynomial. If $S(t)$ is irreducible and $\deg S > 1$, then $S(t)$ will not divide t^j for any j, as shown in the preceding paragraph. Thus, Corollary 6.18 implies that $c_S(x) = c_S(y)$ if and only if $S(t)$ divides $t^k + 1$, which it must do for some big enough k:

Theorem 6.19 *If $S(t)$ is any mod-2 polynomial with at least two terms, then there is an integer $N > 0$, depending on S, such that $S(t)$ divides $t^N + 1$, but $S(t)$ does not divide $t^k + 1$ for any integer $0 < k < N$.*

Proof: Consider the set $\{t^k \,\%\, S(t) : k = 0, 1, \ldots\}$, which consists of representative mod-2 polynomials of degree less than $\deg S$. Since there are only finitely many of those, there must be integers $p > q \geq 0$ such that $t^p = t^q \pmod{S(t)}$. Since $S(t)$ has at least two terms, it cannot divide t^k for any k, and thus $t^q \neq 0 \pmod{S(t)}$. But then $t^{p-q} = 1 \pmod{S(t)}$, so since $1 + 1 = 0 \pmod 2$ we have found $n \stackrel{\text{def}}{=} p - q > 0$ such that $t^n = 1 \pmod{S(t)}$.

Since the set of positive integers $\{n : t^n = 1 \pmod{S(t)}\}$ is nonempty, it has a smallest element which we may call N. By construction, $S(t)$ does not divide $t^k + 1$ for any integer $0 < k < N$. □

Let N be the minimal positive integer for an irreducible polynomial $S(t)$ with at least two terms. Then CSG-2 is achieved for all strings of N or fewer bits, as they correspond to mod-2 polynomials of degree $N - 1$ or less.

For checksum goals 3 and 4, detecting other bit errors, note that the mod-2 polynomial $S(t)$ yields 2^d different checksums,[10] where $d = \deg S$. Under the assumption that all values of $c_S(x)$ are equally likely with our potential bit strings

[10] Why doesn't it yield $S(2)$ checksums?

x, the probability that two different bit strings have the same checksum is $\epsilon = 2^{-d}$. This is made small by choosing S with large degree d.

CRC-32

The standard 32-bit *cyclic redundancy code*, or *CRC-32*, gives a comprehensive example of checksum principles. It is defined by the mod-2 polynomial

$$S(t) = t^{32}+t^{26}+t^{23}+t^{22}+t^{16}+t^{12}+t^{11}+t^{10}+t^8+t^7+t^5+t^4+t^2+t+1. \quad (6.12)$$

This is represented by the coefficient bitstring 1 0000 0100 1100 0001 0001 1101 1011 0111 (base 2) = 4 374 732 215 (base 10) = 104C11DB7 (base 16). Trial division with the $2^{16} = 65536$ mod-2 polynomials of degree 16 or less shows that $S(t)$ is irreducible. Likewise, after a few billion shift and exclusive-or operations, we can show that $S(t)$ divides $t^N + 1$ for no smaller exponent than $N = 4294967295 = 2^{32} - 1$. Hence, using $S(t)$ as a checksum modulus achieves CSG-1 and CSG-2 for all strings of fewer than 4 gigabits. Degree 32 insures that CSG-4 is achieved with probability $\epsilon = 2^{-32} < 10^{-9}$

The POSIX 1003.2 and ISO 8802-3 standard implementation of the CRC-32 checksum is based on computing the remainder modulo $S(t)$, but it performs a few more steps to achieve CSG-3. The checksum $s = c_S(x)$ of a bitstring x, where $s(t)$ and $x(t)$ are both treated as mod-2 polynomials, satisfies

$$t^{32}\tilde{x}(t) = q(t)S(t) + \tilde{s}(t),$$

where $\tilde{x}$ is derived from x by padding the end with zero bits to get an integral number of 8-bit bytes, then appending this number of bytes in binary to the end of x in the fewest bytes sufficient to encode it. Thus the degree of $\tilde{x}(t)$ is a little greater than $\deg x$. Finally, the checksum s is derived from $\tilde{s}$ by complementing each bit, which of course makes no practical difference.

6.3 Exercises

1. Label the vertices in the two graphs of Figure 6.1, and use your labeling to list the set of edges for each. Then pick a root for the right-side graph, which is a tree, determine the depth, and list the vertices by generation.

2. Construct a prefix code for the alphabet $A = \{a, b, c, d\}$ with codeword lengths 1,2,2,3, or prove that none exists.

3. Suppose we have two prefix codes, $\mathbf{c}_0(a, b, c) = (1, 00, 01)$ and $\mathbf{c}_1(a, b, c) = (0, 10, 11)$, for the alphabet $A = \{a, b, c\}$. Show that the following *dynamic encoding* is uniquely decipherable by finding a decoding algorithm:

Dynamic Encoding Example

```
dynamicencoding( msg[], M ):
[0] Initialize n=0
[1] For m=1 to M, do [2] to [3]
[2]    Transmit msg[m] using code n
[3]    If msg[m]=='c', then toggle n = 1-n
```

(This encoding is called dynamic because the codeword for a letter might change as a message is encoded, in contrast with the *static encodings* studied in this chapter. It gives an example of a uniquely decipherable and instantaneous code which is nevertheless not a prefix code.)

4. Suppose that $p^s = p(x_1 x_2 \cdots x_s) \stackrel{\text{def}}{=} p(x_1)p(x_2)\cdots p(x_s)$ is a probability function on A^s derived from the probability function p on A. Show that $H(p^s) = sH(p)$.

5. Count, with proof, the binary trees of depth 3 or less.

6. Suppose $n > 0$. Show that there are infinitely many binary trees with n leaves. (Hint: Consider the case $n = 1$ and linear trees of K daughters between the root and the leaf.)

7. Generalize Huffman's algorithm to 10-ary trees, which would be used to build codes from strings in base 10. (Hint: consider the 3-ary case first.)

8. Construct a Huffman code for the alphabet $A = \{a, b, c, d, e, f, g, h\}$ with occurrence probabilities $p = (.20, .05, .08, .17, .30, .05, .10, .05)$. Compare its bit rate with $H(p)$.

9. Find a Huffman code for the following message, compute its bit rate, and compare it to the entropy of the message:

   > Trustworthy, loyal, helpful, friendly, courteous, kind, obedient, cheerful, thrifty, brave, clean, reverent.

   For simplicity, ignore all spaces, punctuation marks, and capitalization.

10. Construct a canonical Huffman code for the alphabet and occurrence probabilities of Exercise 8, with the property that no letter has a codeword consisting of just 1-bits. Compute its bit rate.

11. What is the probability of an undetected error in N data bits if they have probability p of being flipped but the parity bit has a possibly different probability q of being flipped?

12. Find a binary code with four 10-bit or shorter codewords, wherein restoration to the nearest codeword corrects any two or fewer bit flips.

13. Prove that casting out elevens will detect all one-digit errors in decimal arithmetic. Find an example one-decimal-digit error undetected by casting out nines.

14. Will the combination of checksums c_9 and c_{11} detect all 2-digit errors in decimal arithmetic?

15. Suppose that x and y are integer-size mod-2 polynomials for your computer. Find an alternate implementation of Euclid's algorithm on page 213 using bitwise integer operations. (Hint: look at Euclid's algorithm for integers on page 4.)

16. Suppose that x, y, z are mod-2 polynomials. Prove that if z divides xy and x, z are relatively prime, then z divides y. (This is Lemma 6.16.)

17. a. Show that the mod-2 polynomial $t^3 + t + 1$ is irreducible.

    b. Find a factorization of the mod-2 polynomial $t^4 + t^2 + 1$.

18. For the following mod-2 polynomials $s(t)$, find the least positive integer N such that $s(t)$ divides $t^N + 1$:

    a. $s(t) = t^3 + t + 1$

    b. $s(t) = t^{12} + t^{11} + t^3 + t^2 + t + 1$.

    c. $s(t) = t^{16} + t^{15} + t^2 + 1$.

    d. $s(t) = t^{24} + t^{23} + t^{14} + t^{12} + t^8 + 1$.

19. For the CRC-32 mod-2 polynomial $s(t) = t^{32} + t^{26} + t^{23} + t^{22} + t^{16} + t^{12} + t^{11} + t^{10} + t^8 + t^7 + t^5 + t^4 + t^2 + t + 1$, or 100000100110000010001110110110111 (base 2), show that the least positive integer N for which $s(t)$ divides $t^N + 1$ is $N = 4294967295 = 2^{32} - 1$.

20. Encode the text string "Elephants are approaching from the South!" into ASCII, using seven low-order data bits plus one leading parity bit per character to preserve even parity.

21. Using the mod-2 polynomial $S(t) = t^3 + t + 1$, compute the three-bit checksum for the encoding of "Elephants are approaching from the South!" obtained in previous Exercise 20. Do the same for a similar encoding of "Elephants are approaching from the North!" and "Elephants are approaching from the NORTH!" Do the checksums distinguish these messages?

6.4 Further Reading

- Robert B. Ash. *Information Theory.* Dover Publications, New York, 1965 and 1990. ISBN 0-486-66521-6.

- Richard W. Hamming. *Coding and Information Theory.* Prentice-Hall, Englewood Cliffs, New Jersey, second edition, 1986. ISBN 0131390724.

- Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, Englewood Cliffs, New Jersey, third edition, 1996. ISBN 0-13-349945-6.

- Stephen B. Wicker. *Error Control Systems for Digital Communication and Storage.* Prentice-Hall, Upper Saddle River, New Jersey, 1995. ISBN 0-13-200809-2.

- Ross N. Williams. *A Painless Guide to CRC Error Detection Algorithms.* Rocksoft Pty Ltd., Adelaide, South Australia, 1993. Available from URL `ftp://ftp.rocksoft.com/papers/crc_v3.txt`.

This Page Intentionally Left Blank

Appendix A

Answers

A.1 ...to Chapter 1 Exercises

1. **Solution:** If a divides b and b divides a, one can only conclude that $|a| = |b|$. □

2. **Solution:** Translate the pseudocode on page 4:

Standard C Function: Greatest Common Divisor

```
int gcd ( int a, int b ) {
  int c;
  while ( a != 0 ) { c = a; a = b%a;  b = c; }
  return b;
}
```

Note that Euclid's algorithm is recursive: if $a = 0$, then $\gcd(a, b) = b$; otherwise, $\gcd(a, b) = \gcd(b\%a, a)$. Hence it has another implementation:

Recursive Standard C Function: Greatest Common Divisor

```
int gcd ( int a, int b ) {
  if ( a==0 ) return b;
  return gcd ( b%a, a );
}
```

□

3. **Solution:** Suppose p and q are prime with $p \neq q$, and let $c = \gcd(p, q)$. Recall that their definitions require $c > 0$, $p > 1$ and $q > 1$. Since c divides p, either $c = 1$ or $c = p$. Since c also divides q, either $c = 1$ or $c = q$. Since $p \neq q$, the only solution is therefore $c = 1$. □

4. **Solution:** Using trial division, we discover that the numbers factor as follows: $299792458 = (2)(7)(73)(293339)$, $6447287 = (7)(11)(31)(37)(73)$, and $256964964 = (2)(2)(3)(73)(293339)$. The only prime factor common to all three is 73. However, finding these factorizations required $\sqrt{293339} \approx 10^3$ operations, because of the large prime factor 293339.

   Euclid's algorithm finds the solution with far fewer operations. We first compute $\gcd(299792458, 6447287) = 511$, by

$$\begin{aligned} 299792458\%6447287 &= 3217256, \\ 6447287\%3217256 &= 12775, \\ 3217256\%12775 &= 10731, \\ 12775\%10731 &= 2044, \\ 10731\%2044 &= 511, \\ 2044\%511 &= 0. \end{aligned}$$

   Then $\gcd(256964964, 511) = 73$, since

$$\begin{aligned} 256964964\%511 &= 438, \\ 511\%438 &= 73, \\ 438\%73 &= 0. \end{aligned}$$

   Then we use the fact that $\gcd(a, b, c) = \gcd(a, \gcd(b, c))$. □

5. **Solution:** First compute $\gcd(2301, 19687) = 1$, using Euclid's algorithm, to insure by Lemma 1.6 that a quasi-inverse exists. Then, translate the pseudocode on page 8 to compute it:

Standard C Program: Quasi-inverses

```
#include <stdio.h>
int main ( void ) {
  int a, b, x, y, xo, yo, c, d, t, q, r;
  a = 2301;  b = 19687;
  c=a; d=b;  x=0; yo=0;  y=1; xo=1;
  while ( (r=c%d) > 0 ) {
    q=c/d;  c=d;  d=r; t=xo; xo=x;
    x=t-q*x; t=yo; yo=y; y=t-q*y;
  }
  printf("Inputs a=%d, b=%d;\n", a, b );
  printf("Outputs x=%d, y=%d, gcd(a,b)=d=%d\n",
          x, y, d );
  return 0;
}
```

   This prints

```
Inputs a=2301, b=19687;
Outputs x=-3294, y=385, and gcd(a,b)=d=1
```

The quasi-inverse $x = -3294$ should be adjusted by adding 19 687, making it 16 393 and bringing it into the range $\{0, 1, \ldots, 19\,686\}$. □

6. **Solution:** Overflow occurs if and only if the sum of two positive numbers x, y gives the w-bit twos complement representation of a negative number, which will happen if and only if $0 < x < 2^{w-1}$ and $0 < y < 2^{w-1}$, but $2^{w-1} \leq x + y < 2^w$ considered as counting numbers. But then the sign bit, number $w - 1$, of both x and y must be 0, so that $x + y$ causes no carry out from bit $w - 1$ to bit w. However, sign bit $w - 1$ of $x + y$ is 1, which must have come in as a carry.

   Underflow occurs if and only if the sum of two negative numbers x, y gives a w-bit twos complement positive number. This will happen if and only if $2^{w-1} \leq x < 2^w$ and $2^{w-1} \leq y < 2^w$, but $2^w < x + y < 2^w + 2^{w-1}$, where everything is considered a counting number. But then the sign bit, number $w - 1$, of both x and y must be 1, so that $x + y$ causes a carry out from bit $w - 1$ to bit w. However, sign bit $w - 1$ of $x + y$ is 0, which means there cannot have been a carry in to bit $w - 1$ from bit $w - 2$. □

7. **Solution:** 14 600 926 (base 10) equals $DECADE$ (base 16). □

8. **Solution:** If $\sqrt{p}$ were a rational number, we could write $\sqrt{p} = a/b$ in lowest terms, namely using relatively prime $a, b \in \mathbf{Z}$. But then $b^2 p = a^2$, so p divides a^2. By Lemma 1.3, p divides a, so we can write $a = pa_0$ with $a_0 \in \mathbf{Z}$. But then $p = p^2 a_0^2 / b^2$, so $b^2 = pa_0^2$ and consequently p divides b^2. Again by Lemma 1.3, p divides b. Hence a, b share the common divisor $p > 1$, contradicting the hypothesis that they are relatively prime. □

9. **Solution:** Below is a Standard C implementation of the algorithm on page 9:

Standard C Program: Decimal to Hexadecimal and Binary

```
#include <stdio.h>
int main ( void ) {
  int x;
  printf("Enter a positive integer (base 10): ");
  scanf( "%d", &x);
  printf("%d (base 10) = %x (base 16) = (2 esab) ",x,x);
  for ( ; x>0 ; x /= 2 ) putchar( x&1 ? '1' : '0' );
  putchar('\n');
  return 0;
}
```

□

10. **Solution:** The integer part of π, namely 3, will be the first hexadecimal digit. The remaining seven, plus one more for use in rounding, will be brought to the left of the decimal point if we multiply by $16^8 = 2^{32} = 4\,294\,967\,296$ to get $2^{32}\pi \approx 13\,493\,037\,704.5$. Then we use a variation of the program from the solution to Exercise 6:

Standard C Program: Print Eight Hexadecimal Digits of Pi

```
#include <stdio.h>
int main(void) {
  long long int pihex;
  pihex = (long long int)(3.1415926535897932*(1LL<<32));
  printf("Reversed PI: ");
  do { printf("%X ",(int)(pihex%16)); pihex/=16;
  } while(pihex);
  printf("(esab 16)\n");
  return 0;
}
```

Its output is:

```
Reversed PI: 8 8 A 6 F 3 4 2 3 (esab 16)
```

which means that $\pi \approx 3.243F6A88$ (base 16). Since the least significant digit is the middle value 8, both 8-digit hexadecimal approximations $\pi \approx 3.243F6A9$ (base 16) and $\pi \approx 3.243F6A8$ (base 16) are equally close. □

11. **Solution:** The truncation error will be 2^{-52}, or about 2.2×10^{-16} so there are 15 decimal digits of accuracy. □

12. **Solution:** The sum in exact arithmetic is 1.0×10^8, of course, but on the example machine with $\epsilon_f = 1.19209290 \times 10^{-7}$, the following program printed `sum = 1.677722e+07`:

Standard C Program to Add 10^8 Floating-Point 1.0's

```
#include <stdio.h>
int main ( void ) {
  float sum=0;  long i;
  for ( i=0; i<100000000L; i++ ) sum += 1.0;
  printf( "sum = %e\n", sum);  return 0;
}
```

Notice that `sum`, which is approximately $2/\epsilon_f = 1/\epsilon_0$, stopped increasing at a value for which $1/\texttt{sum} = \epsilon_0 < \epsilon_f$. □

13. **Solution:** The following Standard C code reads the bits as a text string:

Standard C Program: Read IEEE 32-bit Floating Point

```
#include <stdio.h>
#include <math.h>
int main ( void ) {
  int s, e, f, power;
  printf("IEEE reader: Enter 32 bits, each 0 or 1:\n");
  s = ( getchar()=='0' ) ? '+' : '-' ; /* sign */
  for( e=0, power=1<<7 ; power>0 ; power/= 2 )
    if( getchar()=='1' ) e += power;   /* exponent */
  for ( f=0, power=1<<22 ; power>0 ; power/=2 )
    if( getchar()=='1' ) f += power;   /* mantissa */
  if( e==255 )
    if( f==0 ) printf("%c infinity", s);
    else printf("NaN");
  else
    if( e==0 )
      if( f==0 ) printf("%c 0", s);
      else printf("%c %.7e (subnormal)",
            s, pow(2.0,-126.0)*(f/(double)(1<<23)) );
    else printf("%c %.7e",
          s, pow(2.0,e-127.0)*(1.0+f/(double)(1<<23)));
  putchar('\n');
  return 0;
}
```

□

14. **Solution:** Let $F(x) = x_1/x_2$, so $\nabla F(x) = (1/x_2, -x_1/x_2^2)$. Thus $M_1 \approx 1/|x_2|$ and $M_2 \approx |x_1|/|x_2^2|$, so the right-hand side of Inequality 1.11 simplifies to

$$\left(1 + \frac{|x_1/x_2| + |x_1/x_2|}{|x_1/x_2|}\right)\epsilon_f = 3\epsilon_f,$$

which is slightly better than Inequality 1.7. □

15. **Solution:** a. Well-conditioned: Let $F(x,y) = \sqrt{x^2+y^2}$, so

$$\nabla F(x,y) = (x/\sqrt{x^2+y^2}, y/\sqrt{x^2+y^2}).$$

The right-hand side of Inequality 1.11 becomes $2\epsilon_f$.

b. Well-conditioned except near $x = 1$: Let $F(x) = x\log x$, so $F'(x) = 1 + \log x$. Note that

$$|xF'(x)/F(x)| = |(x + x\log x)/x\log x| = \left|1 + \frac{1}{\log x}\right|,$$

which is large near $x = 1$ but bounded elsewhere.

c. Ill-conditioned near any integer $x = k$: The function $F(x) = \lfloor x \rfloor$ is discontinuous at each integer k, so for any $\epsilon > 0$ we can find real numbers x, y with $k < x < k + 1$, $k - 1 < y < k$, and $0 < |x - y| < \epsilon$, such that $F(x) = k$ while $F(y) = k - 1$. If $k = 0$, then $F(x) = 0$ and the relative error in $F(x)$ is undefined. If $k \neq 0$, then $|F(x) - F(y)|/|F(x)| = 1 = \frac{1}{\epsilon_f}\epsilon_f$, and $\epsilon_f \gg 1$. □

A.2 ... to Chapter 2 Exercises

1. **Solution:** The 5-cube is created by sweeping the 4-cube through a fifth axis. It has $2^5 = 32$ vertices. We get 64 edges from the front and back 4-cubes, plus 16 new edges joining corresponding vertices on the front and back, for a total of 80.

   A general formula may be developed recursively. Let $v(N), e(N)$ be the number of vertices and edges, respectively, of an N-cube. We may consider the 0-cube to be a single point, so $v(0) = 1$ and $e(0) = 0$. The 1-cube is a unit line segment, so $v(1) = 2$ and $e(1) = 1$. Counting the vertices gives $v(N) = 2^N$, and for the edges we reason as in the previous paragraph to get the relation

   $$e(N+1) = 2e(N) + v(N) = 2e(N) + 2^N, \qquad N \geq 1.$$

   This has the closed form solution $e(N) = N2^{N-1}$, as may be proved by induction. □

2. **Solution:** The zero subspace $\{0\} \subset \mathbf{R}^N$ is zero-dimensional. For $k = 1, \ldots, N$, let $Y_k = \text{span}\,\{\mathbf{e}_1, \ldots, \mathbf{e}_k\}$. Then Y_k is a k-dimensional subspace of $\mathbf{R}^N$. □

3. **Solution:** Fix N and let $\{\mathbf{e}_n\}$ be the standard basis for $\mathbf{C}^N$. Then $\mathbf{e}_1 = (1, 0, \ldots, 0)$ satisfies $\|\mathbf{e}_1\|_1 = \|\mathbf{e}_1\|_2 = \|\mathbf{e}_1\|_\infty = 1$, making the right column into equalities, while $\mathbf{f}_N = \mathbf{e}_1 + \cdots + \mathbf{e}_N = (1, 1 \ldots, 1)$ satisfies $\|\mathbf{f}_N\|_1 = N$, $\|\mathbf{f}_N\|_2 = \sqrt{N}$, and $\|\mathbf{f}_N\|_\infty = 1$, giving equalities in the left column. □

4. **Solution:** (i) $\Rightarrow$ (ii): Given vectors $\mathbf{x}', \mathbf{y}'$ and scalars a, b, substitute $\mathbf{x} \leftarrow a\mathbf{x}'$ and $\mathbf{y} \leftarrow b\mathbf{y}'$ into the triangle inequality $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ from (i), and then expand.

   (ii) $\Rightarrow$ (i): For the triangle inequality, take $a = b = 1$ in (ii). To show that $\|a\mathbf{x}\| = |a|\,\|\mathbf{x}\|$, start by taking $b = 0$ in condition (ii) to get $\|a\mathbf{x}\| \leq |a|\,\|\mathbf{x}\|$ for every scalar a. Then using this inequality with $a = 1 = \frac{1}{c} \cdot c$ shows that for all $c \neq 0$,

   $$\|\mathbf{x}\| = \|\frac{1}{c} \cdot c\mathbf{x}\| \leq \frac{1}{|c|}\,\|c\mathbf{x}\| \quad \Rightarrow \quad |c|\,\|\mathbf{x}\| \leq \|c\mathbf{x}\|.$$

   But $0 = |0|\,\|\mathbf{x}\| \leq \|0\mathbf{x}\| = 0$, so this second inequality $|a|\,\|\mathbf{x}\| \leq \|a\mathbf{x}\|$ actually holds for all scalars a. Combined with the first inequality, it implies that $\|a\mathbf{x}\| = |a|\,\|\mathbf{x}\|$ for all scalars a. □

5. **Solution:** If $\mathbf{x} \in \mathbf{Y}$ and $\mathbf{x} \in \mathbf{Y}^\perp$, then $\langle \mathbf{x}, \mathbf{x} \rangle = 0$, so $\mathbf{x} = 0$ because the inner product is nondegenerate. For the second part, apply definition 2.30: for any $\mathbf{y} \in \mathbf{Y}$, we have $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ for all $\mathbf{x} \in \mathbf{Y}^\perp$, so $\mathbf{Y} \subset \left(\mathbf{Y}^\perp\right)^\perp$. □

6. **Solution:** Any $\mathbf{y} \in \mathbf{Y}$ may be expanded as $\mathbf{y} = \sum_{n=1}^{N} a_n \mathbf{y}_n$ for some scalars $a_1, \ldots, a_N$. Thus $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{n=1}^{N} a_n \langle \mathbf{x}, \mathbf{y}_n \rangle = 0$, since every term is zero by hypothesis. □

7. **Solution:** First note that these three vectors are linearly independent: if $a\mathbf{x} + b\mathbf{y} + c\mathbf{z} = 0$, then $(a+b+c, a, a+b, a+c) = (0,0,0,0)$, which implies $a = b = c = 0$. Applying the recursive Gram-Schmidt construction gives the orthonormal set $\{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$, where

$$\begin{aligned} \mathbf{p} &= \frac{1}{\|\mathbf{x}\|} x = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right); \\ \mathbf{q}' &= \mathbf{y} - \langle \mathbf{p}, \mathbf{y} \rangle \mathbf{p} = \left(\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}\right) = \mathbf{q}; \\ \mathbf{r}' &= \mathbf{z} - \langle \mathbf{p}, \mathbf{z} \rangle \mathbf{p} - \langle \mathbf{q}, \mathbf{z} \rangle \mathbf{q} = \left(\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}\right) = \mathbf{r}, \end{aligned}$$

since both $\mathbf{q}'$ and $\mathbf{r}'$ happen to be unit vectors. □

8. **Solution:** Recall that $\{\mathbf{e}_n : n = 1, \ldots, N\}$ is an orthonormal basis for $\mathbf{E}^N$.

If $i = j$, then $\langle \mathbf{f}_i', \mathbf{f}_i \rangle = \langle \mathbf{e}_i - \mathbf{e}_{i+1}, \mathbf{e}_1 + \cdots + \mathbf{e}_i \rangle = \langle \mathbf{e}_i, \mathbf{e}_i \rangle = 1$.

If $i > j$, then $\langle \mathbf{f}_i', \mathbf{f}_j \rangle = \langle \mathbf{e}_i - \mathbf{e}_{i+1}, \mathbf{e}_1 + \cdots + \mathbf{e}_j \rangle = 0$, since no component of the left factor is present in the right.

If $i < j$, then the left and right factors of the inner product share exactly two components:

$$\langle \mathbf{f}_i', \mathbf{f}_j \rangle = \langle \mathbf{e}_i - \mathbf{e}_{i+1}, \mathbf{e}_1 + \cdots + \mathbf{e}_j \rangle = \langle \mathbf{e}_i, \mathbf{e}_i \rangle + \langle -\mathbf{e}_{i+1}, \mathbf{e}_{i+1} \rangle = 1 - 1 = 0.$$

Thus $\langle \mathbf{f}_i', \mathbf{f}_j \rangle = \delta(i-j)$. □

9. **Solution:** If u and v are continuous functions, then so is $\bar{u}v$, so the Riemann integral of $\bar{u}v$ exists.

Hermitean symmetry follows from the identities $\bar{v}u = \overline{\bar{u}v}$ and $\int_0^1 \bar{w}(t)\,dt = \overline{\int_0^1 w(t)\,dt}$.

Nondegeneracy follows from the fact that if $w = w(t)$ is nonnegative and continuous on $[0,1]$ and satisfies $\int_0^1 w(t)\,dt = 0$, then $w(t) = 0$ for every $t \in [0,1]$. Taking $w(t) = |u(t)|^2$ then shows that $\langle u, u \rangle = 0$ if and only if $u(t) = 0$ at all $t \in [0,1]$.

Linearity follows since $\bar{u}(av + bw) = a\bar{u}v + b\bar{u}w$ for any continuous functions u, v, w on $[0,1]$ and any complex numbers a, b. The linearity of the Riemann integral then implies $\langle u, av + bw \rangle = a \langle u, v \rangle + b \langle u, w \rangle$. □

10. **Solution:** By the triangle inequality, for every $\mathbf{x} \neq \mathbf{0}$ we have

$$\|(S+T)\mathbf{x}\| \leq \|S\mathbf{x}\| + \|T\mathbf{x}\| \leq \|S\|_{\mathrm{op}} \|\mathbf{x}\| + \|T\|_{\mathrm{op}} \|\mathbf{x}\|.$$

Now divide by $\|\mathbf{x}\| > 0$ to get $\|S+T\|_{\text{op}} \leq \|S\|_{\text{op}} + \|T\|_{\text{op}}$.

Since $\|aT\mathbf{x}\| = |a|\,\|T\mathbf{x}\|$ for every $\mathbf{x}$, and for every $\epsilon > 0$ there is some $\mathbf{x}$ with $(\|T\|_{\text{op}} - \epsilon)\|\mathbf{x}\| < \|T\mathbf{x}\| \leq \|T\|_{\text{op}}\|\mathbf{x}\|$, we conclude that for every $\epsilon > 0$ there is some $\mathbf{x}$ with $(|a|\,\|T\|_{\text{op}} - |a|\epsilon)\|\mathbf{x}\| < \|aT\mathbf{x}\| \leq |a|\,\|T\|_{\text{op}}\|\mathbf{x}\|$. But then we must have $\|aT\|_{\text{op}} = |a|\,\|T\|_{\text{op}}$.

To show that $\|ST\|_{\text{op}} \leq \|S\|_{\text{op}}\|T\|_{\text{op}}$, observe that $\|ST\mathbf{x}\| \leq \|S\|_{\text{op}}\|T\mathbf{x}\| \leq \|S\|_{\text{op}}\|T\|_{\text{op}}\|\mathbf{x}\|$. □

11. **Solution:** The Hilbert–Schmidt norm is derived from the Hermitean inner product $\langle A, B\rangle = \sum_{m,n} \overline{A(m,n)}B(m,n)$ on MN-dimensional complex Euclidean space, with $\|A\|_{\text{HS}} = \sqrt{\langle A, A\rangle}$. Hence it is nondegenerate, and it is sublinear by the Minkowski inequality. To prove submultiplicativity, we first expand in matrix coefficients. Since A, B can be composed into AB, we must have $A \in \mathbf{Mat}(M \times K)$ and $B \in \mathbf{Mat}(K \times N)$ for positive integers M, K, N. But then

$$\|AB\|_{\text{HS}}^2 = \sum_{m=1}^{M}\sum_{n=1}^{N}\left|\sum_{k=1}^{K} A(m,k)B(k,n)\right|^2 = \sum_{m,n} |\,\langle A_m^*, B_n\rangle\,|^2,$$

where B_n is the n^{th} column of B and A_m^* is the m^{th} column of A^*, namely the complex conjugate of the m^{th} row of A, and $\langle \mathbf{x}, \mathbf{y}\rangle = \sum_k \bar{x}(k)y(k)$ is the usual Hermitean symmetric inner product on complex Euclidean K-space. By the Cauchy–Schwarz inequality relating this inner product to the norm $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x}\rangle}$, we have

$$|\,\langle A_m^*, B_n\rangle\,|^2 \leq \|A_m^*\|^2\|B_n\|^2 = \left(\sum_k |A(m,k)|^2\right)\left(\sum_{k'} |B(k',n)|^2\right),$$

since $|\bar{z}| = |z|$ for every z. After summing in m and n we get

$$\|AB\|_{\text{HS}}^2 \leq \left(\sum_{m,k} |A(m,k)|^2\right)\left(\sum_{k',n} |B(k',n)|^2\right) = \|A\|_{\text{HS}}^2\|B\|_{\text{HS}}^2.$$

Taking square roots gives the desired inequality. □

12. **Solution:** Write $A = \{A(m,n)\}$ and $B = \{B(m,n)\}$. Substituting $A \leftarrow A^*$ in Equation 2.46 gives

$$\operatorname{tr}(A^*B) = \sum_n\sum_k A^*(n,k)B(k,n) = \sum_n\sum_k \overline{A(k,n)}B(k,n) = \langle A, B\rangle.$$

Equation 2.46 also gives $\operatorname{tr}(BA^*) = \operatorname{tr}(A^*B) = \langle A, B\rangle$. Next, note that $\operatorname{tr}(C^*) = \overline{\operatorname{tr}(C)}$ for any square matrix C. Since $AB^* = (BA^*)^*$ and $B^*A = (A^*B)^*$ are square, we get $\operatorname{tr}(AB^*) = \operatorname{tr}(B^*A) = \overline{\langle A, B\rangle}$.

Finally, substituting $B \leftarrow A$ in the first result yields $\langle A, A\rangle = \operatorname{tr}(A^*A) = \operatorname{tr}(AA^*)$, since $\langle A, A\rangle$ is real-valued, and the second result follows since $\|A\|_{\text{HS}}$ is the derived norm. □

13. **Solution:** First note that $P_{\mathbf{Y}}$ is linear, since the inner product is linear in the second factor. Second, note that $P_{\mathbf{Y}}\,\mathbf{x} \in \operatorname{span}\mathbf{Y}$ for each $\mathbf{x} \in \mathbf{X}$. Third, note that $P_{\mathbf{Y}}\,\mathbf{y}_k = \mathbf{y}_k$ for $k = 1, \ldots, N$, so $\operatorname{span}\mathbf{Y} \subset P_{\mathbf{Y}}\mathbf{X}$. Hence $P_{\mathbf{Y}}$ is a linear transformation of $\mathbf{X}$ onto $\mathbf{Y}$.

For the projection property, use linearity to compute

$$\begin{aligned} P_{\mathbf{Y}}^2\,\mathbf{x} &= P_{\mathbf{Y}}\left(\sum_{k=1}^{N}\langle \mathbf{y}_k, \mathbf{x}\rangle\,\mathbf{y}_k\right) = \sum_{j=1}^{N}\left\langle \mathbf{y}_j, \sum_{k=1}^{N}\langle \mathbf{y}_k, \mathbf{x}\rangle\,\mathbf{y}_k\right\rangle \mathbf{y}_j \\ &= \sum_{j=1}^{N}\sum_{k=1}^{N}\langle \mathbf{y}_k, \mathbf{x}\rangle\,\langle \mathbf{y}_j, \mathbf{y}_k\rangle\,\mathbf{y}_j = \sum_{j=1}^{N}\langle \mathbf{y}_j, \mathbf{x}\rangle\,\mathbf{y}_j = P_{\mathbf{Y}}\,\mathbf{x}, \end{aligned}$$

since $\langle \mathbf{y}_j, \mathbf{y}_k\rangle = 1$ for $k = j$ and is zero otherwise. Thus $P_{\mathbf{Y}}$ is a projection.

To show that it is an orthogonal projection, we show that $P_{\mathbf{Y}}$ is selfadjoint and use Lemma 2.21. But for arbitrary $\mathbf{x}, \mathbf{z} \in \mathbf{X}$,

$$\begin{aligned} \langle \mathbf{z}, P_{\mathbf{Y}}\,\mathbf{x}\rangle &= \left\langle \mathbf{z}, \sum_{k=1}^{N}\langle \mathbf{y}_k, \mathbf{x}\rangle\,\mathbf{y}_k\right\rangle = \sum_{k=1}^{N}\langle \mathbf{y}_k, \mathbf{x}\rangle\,\langle \mathbf{z}, \mathbf{y}_k\rangle \\ &= \sum_{k=1}^{N}\overline{\langle \mathbf{y}_k, \mathbf{z}\rangle}\,\langle \mathbf{y}_k, \mathbf{x}\rangle = \sum_{k=1}^{N}\langle\langle \mathbf{y}_k, \mathbf{z}\rangle\,\mathbf{y}_k, \mathbf{x}\rangle \\ &= \left\langle \sum_{k=1}^{N}\langle \mathbf{y}_k, \mathbf{z}\rangle\,\mathbf{y}_k, \mathbf{x}\right\rangle = \langle P_{\mathbf{Y}}\,\mathbf{z}, \mathbf{x}\rangle, \end{aligned}$$

which shows that $P_{\mathbf{Y}}^* = P_{\mathbf{Y}}$. □

14. **Solution:** a. The set $\mathbf{Y}_N = \{\mathbf{y}_n : n = 1, \ldots, N\}$ is orthonormal in $\mathbf{X}$ for any N, so by Exercise 13, the following is an orthogonal projection onto $\operatorname{span}\mathbf{Y}_N$:

$$P_N\mathbf{x} \stackrel{\text{def}}{=} \sum_{n=1}^{N}\langle \mathbf{y}_n, \mathbf{x}\rangle\,\mathbf{y}_n.$$

Evidently $\mathbf{x}_N = P_N\mathbf{x}$, so $\|\mathbf{x}_N\| \le \|\mathbf{x}\|$, and thus $\|\mathbf{x}_N\|^2 \le \|\mathbf{x}\|^2$, by Lemma 2.20. Finally, we use the orthonormality of $\mathbf{Y}_N$ and the linearity and Hermitean symmetry of the inner product to expand

$$\begin{aligned} \|\mathbf{x}_N\|^2 &= \langle \mathbf{x}_N, \mathbf{x}_N\rangle = \left\langle \sum_{n=1}^{N}\langle \mathbf{y}_n, \mathbf{x}\rangle\,\mathbf{y}_n, \sum_{m=1}^{N}\langle \mathbf{y}_m, \mathbf{x}\rangle\,\mathbf{y}_m\right\rangle \\ &= \sum_{n=1}^{N}\sum_{m=1}^{N}\overline{c(n)}c(m)\,\langle \mathbf{y}_n, \mathbf{y}_m\rangle = \sum_{n=1}^{N}|c(n)|^2. \end{aligned}$$

b. We begin by expanding $\|\mathbf{x}-\mathbf{x}_N\|^2 = \langle \mathbf{x}-\mathbf{x}_N, \mathbf{x}-\mathbf{x}_N\rangle = \|\mathbf{x}\|^2 + \|\mathbf{x}_N\|^2 - \langle \mathbf{x}, \mathbf{x}_N\rangle - \langle \mathbf{x}_N, \mathbf{x}\rangle$. Now let P_N be the orthogonal projection defined in part (a), so $\mathbf{x}_N = P_N\mathbf{x}$. Then

$$\langle \mathbf{x}, \mathbf{x}_N\rangle = \langle \mathbf{x}, P_N\mathbf{x}\rangle = \langle \mathbf{x}, P_N^2\mathbf{x}\rangle = \langle P_N\mathbf{x}, P_N\mathbf{x}\rangle = \langle \mathbf{x}_N, \mathbf{x}_N\rangle = \|\mathbf{x}_N\|^2,$$

and also $\langle \mathbf{x}_N, \mathbf{x}\rangle = \|\mathbf{x}_N\|^2$, by Hermitean symmetry. Substitution for the two inner products in the initial expansion gives

$$\|\mathbf{x}-\mathbf{x}_N\|^2 = \|\mathbf{x}\|^2 - \|\mathbf{x}_N\|^2.$$

Thus if $\|\mathbf{x}-\mathbf{x}_N\| \to 0$ as $N \to \infty$, then $\|\mathbf{x}_N\|^2 \to \|\mathbf{x}\|^2$, as $N \to \infty$. Part (a) then implies convergence of the series:

$$\sum_{n=1}^{\infty} |c(n)|^2 = \lim_{N\to\infty} \sum_{n=1}^{N} |c(n)|^2 = \lim_{N\to\infty} \|\mathbf{x}_N\|^2 = \|\mathbf{x}\|^2.$$

Conversely, if $\sum_{n=1}^{\infty} |c(n)|^2 = \|\mathbf{x}\|^2$, then $\|\mathbf{x}_N\|^2 \to \|\mathbf{x}\|^2$ as $N \to \infty$. Thus $\|\mathbf{x}-\mathbf{x}_N\|^2 \to 0$, and so $\|\mathbf{x}-\mathbf{x}_N\| \to 0$, as $N \to \infty$. □

15. **Solution:** Let $\mathbf{e}_n \in \ell^2$ be the elementary sequence $(0, \ldots, 0, 1, 0, \ldots)$, where the single nonzero coordinate appears at index n. Then $\|\mathbf{e}_n\| = 1$, but $\|T\mathbf{e}_n\| = n$, which grows without bound as $n \to \infty$. Hence there is no finite c satisfying $\|T\mathbf{e}_n\| \leq c\|\mathbf{e}_n\|$ for all n. □

16. **Solution:** To prove the first equality, write $G_{ij}(\theta)G_{ij}(-\theta)$ as

$$\begin{aligned}[Id - (\mathbf{e}_{ii}+\mathbf{e}_{jj}) + \cos\theta\,(\mathbf{e}_{ii}+\mathbf{e}_{jj}) + \sin\theta\,(\mathbf{e}_{ij}-\mathbf{e}_{ji})] \times \\ \times [Id - (\mathbf{e}_{ii}+\mathbf{e}_{jj}) + \cos\theta\,(\mathbf{e}_{ii}+\mathbf{e}_{jj}) - \sin\theta\,(\mathbf{e}_{ij}-\mathbf{e}_{ji})] \\ = Id - (\mathbf{e}_{ii}+\mathbf{e}_{jj}) + \left(\cos^2\theta + \sin^2\theta\right)(\mathbf{e}_{ii}+\mathbf{e}_{jj}) \quad = Id.\end{aligned}$$

The second equality follows from $\sin(-\theta) = -\sin\theta$, and the third follows since $G_{ij}(\theta)$ is real-valued. □

17. **Solution:** Write $L = Id + L'$ for the matrix, where Id is the $N \times N$ multiplicative identity matrix. Id is zero except for ones on the main diagonal, and L' is the matrix of zeroes except for the subdiagonal part of column k. We need to show that $L^{-1} = Id - L'$. But $(Id + L')(Id - L') = Id + L' - L' + L'L' = Id + L'L'$, so it suffices to show that $L'L' = 0$. But that is evident, since $L'(i,j) = 0$ if $j \neq k$ or $i \leq k$, so the product $L'(i,n)L'(n,j)$ is zero for all i, j, n, so $L'L'(i,j) = \sum_n L'(i,n)L'(n,j) = 0$ for all i, j. The result now follows from the uniqueness of inverse matrices. □

18. **Solution:** Suppose A, B are lower-triangular $N \times N$ matrices. Then $A(i,j) = 0$ and $B(i,j) = 0$ if $i < j$. But then $A(i,k)B(k,j) = 0$ if $i < k$ or $k < j$,

and one of these will be true for all $k \in \{1, \ldots, N\}$ if $i < j$, so $AB(i,j) = \sum_k A(i,k)B(k,j) = 0$ for $i < j$, so AB is also lower-triangular.

The proof for upper-triangular A, B is obtained by replacing 'lower' with 'upper' and replacing '$<$' with '$>$' in the previous paragraph. □

19. **Solution:** A Pascal program to perform these calculations and graphical displays is available at the author's web site, `http://www.math.wustl.edu/~victor/software/qube/cube.pas`. □

A.3 ... to Chapter 3 Exercises

1. **Solution:** The 1-periodization of the function $f(x) = e^{-|x|}$, for $0 \le x < 1$, is given by

$$\begin{aligned}
\sum_{k=-\infty}^{\infty} e^{-|x+k|} &= \sum_{k=-\infty}^{-1} e^{x+k} + \sum_{k=0}^{\infty} e^{-x-k} \\
&= e^x \sum_{k=-\infty}^{-1} e^k + e^{-x} \sum_{k=0}^{\infty} e^{-k} \\
&= e^{x-1} \sum_{k=-\infty}^{0} e^k + e^{-x} \sum_{k=0}^{\infty} e^{-k} \\
&= \frac{e^x + e^{1-x}}{e-1}.
\end{aligned}$$

□

2. **Solution:** a. That δ_ϵ and τ_α are linear transformations follows from their pointwise operation:

$$\begin{aligned}
\tau_\alpha[au + bv](t) &= au(t-\alpha) + bv(t-\alpha) = a\tau_\alpha u(t) + b\tau_\alpha v(t); \\
\delta_\epsilon[au + bv](t) &= a\epsilon^{-1/2}u(t/\epsilon) + b\epsilon^{-1/2}v(t/\epsilon) = a\delta_\epsilon u(t) + b\delta_\epsilon v(t).
\end{aligned}$$

To show invertibility, compute

$$\begin{aligned}
\tau_\alpha\tau_{-\alpha}u(t) &= [\tau_{-\alpha}u](t-\alpha) = u([t-\alpha]-(-\alpha)) = u(t); \\
\delta_\epsilon\delta_{1/\epsilon}u(t) &= \epsilon^{-1/2}[\delta_{1/\epsilon}u](t/\epsilon) = \epsilon^{-1/2}(1/\epsilon)^{-1/2}u\left([t/\epsilon]\cdot\frac{1}{(1/\epsilon)}\right) \\
&= u(t).
\end{aligned}$$

Invertibility in the other order is shown by substituting $\alpha \leftarrow -\alpha$ and $\epsilon \leftarrow 1/\epsilon$, respectively.

b. First compute $\delta_\epsilon[\tau_\alpha w](t) = \epsilon^{-1/2}[\tau_\alpha w](t/\epsilon) = \epsilon^{-1/2}w\left(\frac{t}{\epsilon}+\alpha\right)$. In the other order, $\tau_\alpha[\delta_\epsilon w](t) = [\delta_\epsilon w](t-\alpha) = \epsilon^{-1/2}w\left(\frac{t-\alpha}{\epsilon}\right)$.

c. Using parts a and b, we compute $v(t) = \delta_\epsilon^{-1}\tau_\alpha^{-1}u(t) = \epsilon^{1/2}u(\epsilon t+\alpha)$. Thus, we may compute $\tau_\alpha\delta_\epsilon F\delta_\epsilon^{-1}\tau_\alpha^{-1}u(t)$ from Equation 3.14 as follows:

$$\tau_\alpha\delta_\epsilon Fv(t) = \epsilon^{-1/2} \times \begin{cases} r\left(\frac{t-\alpha}{\epsilon}\right)v\left(\frac{t-\alpha}{\epsilon}\right) + r\left(\frac{\alpha-t}{\epsilon}\right)v\left(\frac{\alpha-t}{\epsilon}\right), & \text{if } \frac{t-\alpha}{\epsilon} > 0, \\ \bar{r}\left(\frac{\alpha-t}{\epsilon}\right)v\left(\frac{t-\alpha}{\epsilon}\right) - \bar{r}\left(\frac{t-\alpha}{\epsilon}\right)v\left(\frac{\alpha-t}{\epsilon}\right), & \text{if } \frac{t-\alpha}{\epsilon} < 0, \\ v(-\alpha/\epsilon), & \text{if } \frac{t-\alpha}{\epsilon} = 0, \end{cases}$$

Now note that $\epsilon^{-1/2}v\left(\frac{t-\alpha}{\epsilon}\right) = u(t)$, while $\epsilon^{-1/2}v\left(\frac{\alpha-t}{\epsilon}\right) = u(2\alpha-t)$, for every $t \in \mathbf{R}$. Combining this with a simplification of the range limits gives:

$$\tau_\alpha\delta_\epsilon\, F\, \delta_\epsilon^{-1}\tau_\alpha^{-1}u(t) = \begin{cases} r\left(\frac{t-\alpha}{\epsilon}\right)u(t) + r\left(\frac{\alpha-t}{\epsilon}\right)u(2\alpha-t), & \text{if } t > \alpha, \\ \bar{r}\left(\frac{\alpha-t}{\epsilon}\right)u(t) - \bar{r}\left(\frac{t-\alpha}{\epsilon}\right)u(2\alpha-t), & \text{if } t < \alpha, \\ u(\alpha), & \text{if } t = \alpha. \end{cases}$$

Finally, observe that

$$r\left(\frac{t-\alpha}{\epsilon}\right) = \begin{cases} 1, & \text{if } t \geq \alpha+\epsilon, \\ 0, & \text{if } t \leq \alpha-\epsilon, \end{cases} \qquad r\left(\frac{\alpha-t}{\epsilon}\right) = \begin{cases} 0, & \text{if } t \geq \alpha+\epsilon, \\ 1, & \text{if } t \leq \alpha-\epsilon. \end{cases}$$

These valuations result in the formula in Equation 3.20.

d. Performing the previous calculation, but starting from Equation 3.15, we get

$$\begin{aligned} S(r,\alpha,\epsilon)u(t) &= \tau_\alpha \delta_\epsilon \, S \, \delta_\epsilon^{-1} \tau_\alpha^{-1} u(t) \\ &= \begin{cases} \bar{r}(\frac{t-\alpha}{\epsilon})u(t) - r(\frac{\alpha-t}{\epsilon})u(2\alpha - t), & \text{if } \alpha < t \leq \alpha+\epsilon, \\ r(\frac{\alpha-t}{\epsilon})u(t) + \bar{r}(\frac{t-\alpha}{\epsilon})u(2\alpha - t), & \text{if } \alpha-\epsilon \leq t < \alpha, \\ u(t), & \text{otherwise.} \end{cases} \end{aligned}$$

Finally, since the inverse of $\tau_\alpha \delta_\epsilon$ is $\delta_\epsilon^{-1}\tau_\alpha^{-1} = \delta_{1/\epsilon}\tau_{-\alpha}$, we can simplify

$$\begin{aligned} S(r,\alpha,\epsilon)F(r,\alpha,\epsilon) &= \left(\tau_\alpha \delta_\epsilon \, S \, \delta_\epsilon^{-1}\tau_\alpha^{-1}\right)\left(\tau_\alpha \delta_\epsilon \, F \, \delta_\epsilon^{-1}\tau_\alpha^{-1}\right) \\ &= \tau_\alpha \delta_\epsilon \, SF \, \delta_\epsilon^{-1}\tau_\alpha^{-1} \;=\; \tau_\alpha \delta_\epsilon \delta_\epsilon^{-1}\tau_\alpha^{-1} \qquad = Id, \end{aligned}$$

since $SF = Id$. The other order, $F(r,\alpha,\epsilon)S(r,\alpha,\epsilon) = Id$, likewise follows from $FS = Id$. □

3. **Solution:** Suppose that r is a rising cut-off function and $\epsilon < T/2$. Write $F_0 = F(r,0,\epsilon)$ and $F_T = F(r,T,\epsilon)$ for the fraying operators at 0 and T, respectively. Since $u(t+T) = u(t)$ for all t, the function $v \stackrel{\text{def}}{=} F_0 F_T u$ satisfies $v(t+T) = v(t)$ for all $-\frac{T}{2} \leq t \leq \frac{T}{2}$. In particular, that means $v^I(t) = v(t)$ for all $-\epsilon \leq t \leq T+\epsilon$. Also, $v(t) = u(t)$ if t lies between ϵ and $T-\epsilon$, outside the reach intervals of F_0 and F_T.

Now put $I = [0,T]$. The second formula for the loop operator, Equation 3.23, simplifies to

$$L_I v(t) = \begin{cases} S(r,0,\epsilon)v(t), & \text{if } 0 < t \leq \epsilon; \\ S(r,T,\epsilon)v(t), & \text{if } T-\epsilon \leq t < T; \\ v(t), & \text{otherwise.} \end{cases}$$

Writing $S_0 = S(r,0,\epsilon)$ and $S_T = S(r,T,\epsilon)$ for the splicing operators at 0 and T, one calculates

$$L_I v(t) = \begin{cases} S_0 F_0 u(t) = u(t), & \text{if } 0 < t \leq \epsilon, \\ S_T F_T u(t) = u(t), & \text{if } T-\epsilon \leq t < T, \\ v(t) = u(t), & \text{if } \epsilon \leq t \leq T-\epsilon. \end{cases}$$

Thus $L_I F_0 F_T u = u$ on I, so $(L_I F_0 F_T u)^I = u^I = u$ on $\mathbf{R}$. □

4. **Solution:** (i) $t > 1$ implies $\tilde{F}u(t) = \bar{1}u(t) = u(t)$ and $\tilde{S}u(t) = 1u(t) = u(t)$. Likewise, $t < -1$ implies $\tilde{F}u(t) = 1u(t) = u(t)$ and $\tilde{S}u(t) = \bar{1}u(t) = u(t)$.

The rest may be shown directly by modifying Equations 3.18 and 3.19 and the proof of Lemma 3.4. However, notice that $\tilde{F} = RFR$ and $\tilde{S} = RSR$, where the *reflection* R is defined by $Ru(t) = u(-t)$:

$$\begin{aligned} RFRu(t) &= \begin{cases} r(-t)Ru(-t) + r(t)Ru(t), & \text{if } -t > 0, \\ \bar{r}(t)Ru(-t) - \bar{r}(-t)Ru(t), & \text{if } -t < 0, \\ u(0), & \text{if } t = 0, \end{cases} \\ &= \begin{cases} r(-t)u(t) + r(t)u(-t), & \text{if } t < 0, \\ \bar{r}(t)u(t) - \bar{r}(-t)u(-t), & \text{if } t > 0, \\ u(0), & \text{if } t = 0, \end{cases} \\ &= \tilde{F}u(t); \end{aligned}$$

$$\begin{aligned} RSRu(t) &= \begin{cases} \bar{r}(-t)Ru(-t) - r(t)Ru(t), & \text{if } -t > 0, \\ r(t)Ru(-t) + \bar{r}(-t)Ru(t), & \text{if } -t < 0, \\ u(0), & \text{if } t = 0, \end{cases} \\ &= \begin{cases} \bar{r}(-t)u(t) - r(t)u(-t), & \text{if } t < 0, \\ r(t)u(t) + \bar{r}(-t)u(-t), & \text{if } t > 0, \\ u(0), & \text{if } t = 0, \end{cases} \\ &= \tilde{S}u(t). \end{aligned}$$

(ii) R is a linear transformation, since $R[au + bv](t) = au(-t) + bv(-t) = aRu(t) + bRv(t)$. Since F, S are also linear transformations, the compositions $\tilde{F} = RFR$ and $\tilde{S} = RSR$ are linear transformations as well.

(iii) $R^2 = Id$, so $\tilde{F}\tilde{S} = (RFR)(RSR) = RFR^2SR = RFSR = R^2 = Id$, and similarly, $\tilde{S}\tilde{F} = (RSR)(RFR) = Id$.

(iv) By the chain rule, Ru has the same smoothness as u, with $[Ru]^{(n)}(t) = (-1)^n u^{(n)}(-t)$. Thus for odd n it is

$$[\tilde{F}u]^{(n)}(0-) = [RFRu]^{(n)}(0-) = (-1)^n [FRu]^{(n)}(0+) = 0,$$

since by Lemma 3.4, $[FRu]^{(n)}(0+) = 0$ for odd n. Likewise, $[\tilde{F}u]^{(n)}(0+) = [RFRu]^{(n)}(0+) = (-1)^n [FRu]^{(n)}(0-) = 0$ for even n.

(v) Note that $[Ru]^{(n)}(0+) = -u^{(n)}(0-) = 0$ for odd n, and $[Ru]^{(n)}(0-) = u^{(n)}(0+) = 0$ for even n. Lemma 3.4 implies that SRu belongs to C^d, but then so does its reflection $RSRu = \tilde{S}u$.

Finally, observe that r is a rising cut-off function if and only if $\bar{r}$ is, so that replacing $r \leftarrow \bar{r}$ in the definitions of F and S, or $\tilde{F}$ and $\tilde{S}$, gives two more fraying-splicing pairs. Note too that if $r = \bar{r}$ then $\tilde{F} = S$ and $\tilde{S} = F$, so that fraying and splicing are the same except for a reflection of t. □

5. **Solution:** Start by proving that if all the higher-order derivatives of $f = f(t)$ exist and are bounded at all $t \neq \pm 1$, and f has k vanishing derivatives at

$t = \pm 1$ and also satisfies $f(-1) = 0$, $f(1) = 1$, then

$$g(t) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } t \le -1, \\ f\left(\sin \frac{\pi}{2} t\right), & \text{if } -1 < t < 1, \\ 1, & \text{if } t \ge 1, \end{cases}$$

is also smooth for $t \neq \pm 1$, but has $2k+1$ vanishing derivatives at $t = 1$ and $t = -1$.

It is clear that g is as smooth as f off $t = \pm 1$, and that the one-sided limits still match up: $g(-1^-) = 0 = g(-1^+)$ and $g(+1^+) = 1 = g(+1^-)$. Also, all one-sided derivatives over the constant parts vanish: $g^{(n)}(-1^-) = 0 = g^{(n)}(+1^+)$ for all $n = 1, 2, \ldots$. Hence, it remains to evaluate $2k+1$ one-sided derivatives over the nonconstant parts of g near ± 1. This will be done by induction on k.

If $k = 0$, then $\frac{d}{dt} g(\pm 1^{\mp}) = \frac{d}{dt} f(\sin \frac{\pi}{2} t)|_{\pm 1^{\mp}} = \frac{\pi}{2} \frac{df}{dt}|_{\pm 1^{\mp}} \cdot \cos(\pm \frac{\pi}{2}) = 0$, so g has one continuous derivative.

Now suppose that any function with $n < k$ vanishing derivatives at ± 1 gives a g with $2n+1$ vanishing derivatives there. A function f with k vanishing derivatives certainly has $k - 1 < k$ of them, so the first $2k-1$ derivatives $g'(t), \ldots, g^{(2k-1)}(t)$ vanish at $t = \pm 1^{\mp}$. It remains only to show that $g^{(2k)}(\pm 1^{\mp}) = 0$ and $g^{(2k+1)}(\pm 1^{\mp}) = 0$ as well.

But

$$\begin{aligned} \left(\frac{d}{dt}\right)^{2k} f(\sin \frac{\pi}{2} t) &= \left(\frac{d}{dt}\right)^{2k-1} \left[f'(\sin \frac{\pi}{2} t) \cdot \frac{\pi}{2} \cdot \cos \frac{\pi}{2} t \right] \\ &= \sum_{j=0}^{2k-1} \binom{2k-1}{j} \frac{\pi}{2} \left[\left(\frac{d}{dt}\right)^{j} \cos \frac{\pi}{2} t \right] \times \\ &\qquad \times \left[\left(\frac{d}{dt}\right)^{2k-1-j} f'(\sin \frac{\pi}{2} t) \right]. \end{aligned}$$

Now f' has $k-1$ vanishing derivatives at ± 1, so by the induction hypothesis, the last factor in the sum vanishes as $t \to \pm 1$, for all $j = 0, 1, \ldots, 2k-1$. Thus $g^{(2k)}(\pm 1^{\mp}) = 0$.

Likewise, using the fact that the first $2k-1$ derivatives of $f'(\sin \frac{\pi}{2} t)$ vanish at ± 1, the following sum simplifies to the $j = 0$ term:

$$\begin{aligned} \left(\frac{d}{dt}\right)^{2k+1} g(t) &= \sum_{j=0}^{2k} \binom{2k}{j} \frac{\pi}{2} \left[\left(\frac{d}{dt}\right)^{j} \cos \frac{\pi}{2} t \right] \times \\ &\qquad \times \left[\left(\frac{d}{dt}\right)^{2k-j} f'(\sin \frac{\pi}{2} t) \right] \\ &= \frac{\pi}{2} \left[\cos \frac{\pi}{2} t \right] \left(\frac{d}{dt}\right)^{2k-j} f'(\sin \frac{\pi}{2} t). \end{aligned}$$

But $\cos \frac{\pi}{2} t \to 0$ as $t \to \pm 1$, so $g^{(2k+1)}(\pm 1) = 0$, completing the induction.

Finally, apply the result just proved to $r_n(t)$. Let $v(n)$ be the number of vanishing derivatives of $r_n(t)$ at $t = \pm 1$. This is also the number of continuous derivatives of r_n, since it is infinitely smooth with bounded derivatives at all t strictly inside or strictly outside $(-1, 1)$. But $v(0) = 0$, since $r_0(t)$ is merely continuous at $t = -1$ (though it has one continuous derivative at $t = 1$), and $v(n) = 2v(n) + 1$ for $n = 1, 2, \ldots$. This recurrence is solved by $v(n) = 2^n - 1$, proving the result. □

6. **Solution:** First note that by 1-periodicity,

$$\int_0^1 \sin^2 2\pi t \, dt = \int_{\frac{1}{2}}^{\frac{3}{2}} \sin^2 2\pi t \, dt = \int_0^1 \cos^2 2\pi t \, dt.$$

Thus, integrating $\sin^2 \theta + \cos^2 \theta = 1$ over one period is the same as computing $2 \int_0^1 \sin^2 2\pi t \, dt = \int_0^1 1 = 1$, proving Equation 3.66.

Second, recall that $2 \sin A \sin B = \cos(A - B) - \cos(A + B)$. Thus, if n and m are nonnegative with $n \neq m$, then $n - m \neq 0$ and $n + m \neq 0$, so by calculus,

$$\begin{aligned} \left\langle \sqrt{2} \sin 2\pi n t, \sqrt{2} \sin 2\pi m t \right\rangle &= \\ &= 2 \int_0^1 \cos 2\pi (n - m) t \, dt - 2 \int_0^1 \cos 2\pi (n + m) t \, dt \\ &= \frac{\sin 2\pi (n - m) t}{\pi (n - m) t} \Big|_0^1 - \frac{\sin 2\pi (n + m) t}{\pi (n + m) t} \Big|_0^1 = 0, \end{aligned}$$

proving Equation 3.65. An almost identical argument from the identity $2 \cos A \cos B = \cos(A - B) + \cos(A + B)$ proves Equation 3.64.

Finally, recall that $2 \sin A \cos B = \sin(A - B) + \sin(A + B)$. Hence for any nonnegative n and m,

$$\left\langle \sqrt{2} \sin 2\pi n t, \sqrt{2} \cos 2\pi m t \right\rangle = 2 \int_0^1 [\sin 2\pi (n - m) t - \sin 2\pi (n + m) t] \, dt.$$

But $\int_0^1 \sin 2\pi k t \, dt = 0$ for every integer k, in particular for $k = n - m$ or $k = n + m$, so both of the integrals are zero, proving Equation 3.63. □

7. **Solution:** Since $\cos^2 \theta = \frac{1}{2} + \frac{1}{2} \cos 2\theta$, the Fourier series

$$f(x) = a(0) + \sqrt{2} \sum_{n=1}^{\infty} [a(n) \cos(2\pi n x) + b(n) \sin(2\pi n x)]$$

has just two nonzero terms: $a(0) = \frac{1}{2\sqrt{2}}$ and $a(2) = \frac{1}{2\sqrt{2}}$. □

8. **Solution:** Note that $\sin(2\pi kt - d) = \sin(2\pi kt)\cos(d) - \cos(2\pi kt)\sin(d)$. Hence its real Fourier series has $a(k) = -\sin(d)/\sqrt{2}$ and $b(k) = \cos(d)/\sqrt{2}$, with $a(n) = b(n) = 0$ for all $n \neq k$. This can be converted to a complex Fourier series $\sum_n c(n)e^{2\pi int}$ with the formula

$$c(k) = [a(k) - ib(k)]/\sqrt{2}; \qquad c(-k) = [a(k) + ib(k)]/\sqrt{2}; \qquad c(n) = 0,$$

for all $n \neq \pm k$. Substituting gives $c(k) = \frac{-i}{2}e^{-id}$ and $c(-k) = \frac{i}{2}e^{id}$. □

9. **Solution:** The series $f(t) = \sum_{n\in\mathbf{Z}} c(n)e^{2\pi int}$ converges absolutely and uniformly at every t, by comparison with the absolutely convergent series $\sum_{n\neq 0} |n|^{-3}$ that does not depend on t. Likewise, differentiating each term of the sum gives a series $g(t) = 2\pi i \sum_{n\in\mathbf{Z}} nc(n)e^{2\pi int}$ which is uniformly absolutely convergent by comparison with $\sum_{n\neq 0} |n|^{-2}$. Thus $g = f'$ by Apostol, Theorem 9.14, page 230. □

10. **Solution:** If $n \neq m$, then $\phi_n(t)\phi_m(t) = 0$ for every $t \in [0, 1]$, since one of the factors must be zero. The collection $\{\phi_k : 0 \leq k < N\}$ can be made orthonormal by defining $\phi_k(t) = \sqrt{N}\mathbf{1}(Nt - k)$. Its span is the step functions defined on $[0, 1)$ which are constant on subintervals $[\frac{k}{N}, \frac{k+1}{N})$ for $0 \leq k < N$. □

11. **Solution:** Make the change of variable $x \leftarrow y + k$, so $dx \leftarrow dy$, and extract the factor $e^{-2\pi ik\xi}$:

$$\begin{aligned}\mathcal{F}\phi_k(\xi) &= \int_{-\infty}^{\infty} e^{-2\pi ix\xi}\,\phi(x-k)\,dx \\ &= e^{-2\pi ik\xi}\int_{-\infty}^{\infty} e^{-2\pi iy\xi}\,\phi(y)\,dy \quad = e^{-2\pi ik\xi}\mathcal{F}\phi(\xi).\end{aligned}$$

□

12. **Solution:** Make the change of variable $x \leftarrow ay$, so $dx \leftarrow ady$, and extract the factor a:

$$\mathcal{F}\phi_a(\xi) = \int_{-\infty}^{\infty} e^{-2\pi ix\xi}\,\phi(x/a)\,dx = a\int_{-\infty}^{\infty} e^{-2\pi iy(a\xi)}\,\phi(y)\,dy = a\mathcal{F}\phi(a\xi).$$

□

13. **Solution:** From Exercise 12, with $f_a(x) \stackrel{\text{def}}{=} f(x/a)$, we get the identity

$$\mathcal{F}^{-1}f_a(\xi) = \mathcal{F}f_a(-\xi) = a\mathcal{F}f(-a\xi) = a\mathcal{F}^{-1}f(a\xi).$$

Thus, using the hint with the definition $\mathcal{F}\mathbf{1}_I = \text{sinc}$, we calculate

$$\mathcal{F}^{-1}\psi(x) = 2\mathcal{F}\mathbf{1}_I(2x) - \mathcal{F}\mathbf{1}_I(x) = 2\text{sinc}\,(2x) - \text{sinc}\,(x).$$

□

14. **Solution:** First note that integration by parts yields

$$\int_a^b xe^{cx}\,dx = x\frac{e^{cx}}{c}\Big|_a^b - \int_a^b \frac{e^{cx}}{c}\,dx = \frac{be^{cb}-ae^{ca}}{c} - \frac{e^{cb}-e^{ca}}{c^2}.$$

Thus the Fourier integral transform of the hat function $h(x)$ is:

$$\begin{aligned}
\hat{h}(\xi) &= \int_{-\infty}^{\infty} h(x)e^{-2\pi ix\xi}\,dx = \int_{-1}^{1} h(x)e^{-2\pi ix\xi}\,dx \\
&= \int_{-1}^{0}(1+x)e^{-2\pi ix\xi}\,dx + \int_{0}^{1}(1-x)e^{-2\pi ix\xi}\,dx \\
&= \int_{-1}^{1} e^{-2\pi ix\xi}\,dx + \int_{-1}^{0} xe^{-2\pi ix\xi}\,dx - \int_{0}^{1} xe^{-2\pi ix\xi}\,dx \\
&= \frac{e^{-2\pi i\xi}-e^{2\pi i\xi}}{-2\pi i\xi} + \frac{e^{2\pi i\xi}}{-2\pi i\xi} - \frac{1-e^{2\pi i\xi}}{(-2\pi i\xi)^2} - \frac{e^{-2\pi i\xi}}{-2\pi i\xi} + \frac{e^{-2\pi i\xi}-1}{(-2\pi i\xi)^2} \\
&= \frac{e^{2\pi i\xi}+e^{-2\pi i\xi}-2}{(2\pi i\xi)^2} = \left(\frac{e^{\pi i\xi}-e^{-\pi i\xi}}{2\pi i\xi}\right)^2 = \left(\frac{\sin\pi\xi}{\pi\xi}\right)^2.
\end{aligned}$$

But this last is just $\operatorname{sinc}(\xi)^2$. □

15. **Solution:** Use the geometric series summation formula:

$$\begin{aligned}
\sum_{k=0}^{N-1} \overline{\omega_n}(k)\omega_m(k) &= \frac{1}{N}\sum_{k=0}^{N-1} e^{-\frac{2\pi ink}{N}} e^{\frac{2\pi imk}{N}} = \frac{1}{N}\sum_{k=0}^{N-1} e^{\frac{2\pi i(m-n)k}{N}} \\
&= \begin{cases} \frac{1}{N}\frac{1-e^{2\pi i(m-n)}}{1-e^{2\pi i\frac{m-n}{N}}}, & \text{if } n \neq m; \\ \frac{1}{N}N, & \text{if } n = m, \end{cases} \quad = \delta(n-m).
\end{aligned}$$

□

16. **Solution:**

$$F_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}; \qquad F_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}.$$

$$H_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}; \qquad H_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}.$$

□

17. **Solution:** Fix $N > 0$. Then

$$\begin{aligned} F_N^2(n,j) &= \frac{1}{N}\sum_{k=0}^{N-1} \omega_n^k \omega_k^j = \frac{1}{N}\sum_{k=0}^{N-1} e^{-2\pi i(j+n)\frac{k}{N}} \\ &= \begin{cases} 0, & \text{if } j+n \neq 0 \text{ and } j+n \neq N; \\ 1, & \text{if } j+n = N \text{ or } j+n = 0. \end{cases} \end{aligned}$$

The resulting matrix has ones at $(0,0)$ and along the antidiagonal $(n, N-n)$, for $n = 1, 2, \ldots, N-1$, with zeroes at all other locations. □

18. **Solution:** Fix $N > 0$. By the previous calculation, for any vector $v = (v_0, v_1, \ldots, v_{N-1})^T$,

$$F_N^2 v = F_N^2 \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix} = \begin{pmatrix} v_0 \\ v_{N-1} \\ v_{N-2} \\ \vdots \\ v_1 \end{pmatrix}.$$

Thus,

$$F_N^4 v = F_N^4 \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix} = \left(F_N^2\right)^2 \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix} = F_N^2 \begin{pmatrix} v_0 \\ v_{N-1} \\ v_{N-2} \\ \vdots \\ v_1 \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix}.$$

That is, F_N^4 is the $N \times N$ identity matrix. □

19. **Solution:** C_2^{IV} and C_4^{IV} are

$$\begin{pmatrix} \cos\frac{\pi}{8} & \cos\frac{3\pi}{8} \\ \cos\frac{3\pi}{8} & \cos\frac{9\pi}{8} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \cos\frac{\pi}{16} & \cos\frac{3\pi}{16} & \cos\frac{5\pi}{16} & \cos\frac{7\pi}{16} \\ \cos\frac{3\pi}{16} & \cos\frac{9\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{21\pi}{16} \\ \cos\frac{5\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{25\pi}{16} & \cos\frac{35\pi}{16} \\ \cos\frac{7\pi}{16} & \cos\frac{21\pi}{16} & \cos\frac{35\pi}{16} & \cos\frac{49\pi}{16} \end{pmatrix},$$

respectively. □

20. **Solution:** Split the sum for $H_N v(m)$ into its odd and even parts:

$$\begin{aligned} &\sum_{n=0}^{N-1} v(n)\,\mathrm{cas}\,\frac{2\pi mn}{N} = \\ &= \sum_{n=0}^{\frac{N}{2}-1} v(2n)\,\mathrm{cas}\,\frac{2\pi m(2n)}{N} + \sum_{n=0}^{\frac{N}{2}-1} v(2n+1)\,\mathrm{cas}\,\frac{2\pi m(2n+1)}{N} \end{aligned}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} v(2n) \operatorname{cas} \frac{2\pi mn}{N/2} + \sum_{n=0}^{\frac{N}{2}-1} v(2n+1) \operatorname{cas} \left[\frac{2\pi mn}{N/2} + \frac{2\pi m}{N} \right].$$

Now $\operatorname{cas}(A+B) = \cos B \operatorname{cas} A + \sin B \operatorname{cas}(-A)$. Putting $A \leftarrow \frac{2\pi mn}{N/2}$ and $B \leftarrow \frac{2\pi m}{N}$ gives:

$$H_N v(m) = H_{\frac{N}{2}} v^e(m) + \cos \frac{2\pi m}{N} H_{\frac{N}{2}} v^o(m) + \sin \frac{2\pi m}{N} H_{\frac{N}{2}} v^o(-m),$$

where v^e and v^o are the even- and odd-indexed parts of v, respectively. Finally, we use the $\frac{N}{2}$-periodicity of $H_{\frac{N}{2}} v^e(m)$ and $H_{\frac{N}{2}} v^o(m)$ as needed for indices $m = \frac{N}{2}, \ldots, N-1$, and to write $H_{\frac{N}{2}} v^o(-m) = H_{\frac{N}{2}} v^o(\frac{N}{2} - m) = H_{\frac{N}{2}} v^o(N-m)$. □

21. **Solution:** Symmetry is evident, since the (m,n) entry is unchanged when m and n are swapped. To show orthogonality, it remains to establish that for any $0 \le m, n \le N-1$,

$$\sum_{k=0}^{N-1} \sin \frac{\pi(n+\frac{1}{2})(k+\frac{1}{2})}{N} \sin \frac{\pi(k+\frac{1}{2})(m+\frac{1}{2})}{N} = \begin{cases} \frac{N}{2}, & \text{if } m = n, \\ 0, & \text{if } m \neq n. \end{cases}$$

But $2 \sin A \sin B = \cos(A-B) - \cos(A+B)$, so the sum can be rewritten as $I - II$, in the notation of the proof of Theorem 3.15. But if $m \neq n$, then $I = II = 0$ as before. On the other hand, if $m = n$, then the equation holds if

$$\sum_{k=0}^{N-1} \sin^2 \frac{\pi(n+\frac{1}{2})(k+\frac{1}{2})}{N} = \frac{N}{2}.$$

Using Equation 3.59 and the identity $\sin^2\theta + \cos^2\theta = 1$ yields

$$\sum_{k=0}^{N-1} \sin^2 \frac{\pi(n+\frac{1}{2})(k+\frac{1}{2})}{N} = \frac{1}{2} \sum_{k=0}^{N-1} 1 = \frac{N}{2},$$

as in the previous proof. □

A.4 ... to Chapter 4 Exercises

1. **Solution:** The Lagrange polynomial through the points $(0,0)$, $(1,2)$, $(2,4)$ is $\Lambda_2(x) = 2x$. This is a degenerate case of three collinear points; it yields an interpolating polynomial of degree 1. □

2. **Solution:** The Lagrange polynomial is

$$\Lambda_2(x) = q - \frac{p\,x}{2} + \frac{r\,x}{2} + \frac{p\,x^2}{2} - q\,x^2 + \frac{r\,x^2}{2}$$

Its derivative with respect to q is $\frac{d}{dq}\Lambda_2(x) = 1 - x^2$. □

3. **Solution:** a. We use equispaced samples $x_i = i$ with $h = 1$ in Equation 4.6 to get the Newton form of Λ_k. We claim that $f(j) = 2^j$ gives the differences $\Delta^j f(j) = 1$ for all $j = 0, 1, \ldots, k$. This follows from a stronger statement, $\Delta^i f(j) = 2^{j-i}$ for all $i = 0, 1, \ldots, j$, which we will prove by induction on i. But $1 = \Delta^0 f(j) = f(j) = 2^j$ proves the $i = 0$ case, and we get the inductive step from the recursive formula: $\Delta^i f(j) = 2^{j-i}$ implies $\Delta^{i+1} f(j) = \Delta^i f(j) - \Delta^i f(j-1) = 2^{j-i} - 2^{j-1-i} = 2^{j-1-i} = 2^{j-(i+1)}$.

   b. $\binom{n}{j} = \frac{1}{j!}(n-0)(n-1)\cdots(n-[j-1])$ is just the j^{th} term of $\Lambda_k(x)$, written in Newton form as in Equation 4.6 and evaluated at $x = n$.

   c. Since Λ_k is a polynomial of degree k or less, we have $\Lambda_k(n) = O(n^k)$ as $n \to \infty$. In fact, the coefficient of x^k in $\Lambda_k(x)$ is $1/k!$, so $\deg \Lambda_k = k$. □

4. **Solution:** Since f itself is a polynomial of degree 2, it equals its Chebyshev polynomial expansion: $f(x) = c_0 T_0(x) + c_1 T_1(x) + c_2 T_2(x)$, for all $x \in [-1, 1]$. The expansion coefficients may thus be found by the method of undetermined coefficients. But $T_0(x) = 1$, $T_1(x) = x$, and $T_2(x) = 2x^2 - 1$, so:

$$1 - x^2 = c_0 \cdot 1 + c_1 \cdot x + c_2 \cdot (2x^2 - 1) = [c_0 - c_2] + [c_1]x + 2x^2[c_2],$$

   so $c_1 = 0$, $c_2 = -\frac{1}{2}$, and $c_0 = \frac{1}{2}$. □

5. **Solution:** Pick two sample points $x_{k-1} < x_k$. Then $y_{k-1} = mx_{k-1} + b$ and $y_k = mx_k + b$, so the piecewise linear function approximating f on the interval $x_{k-1} \le x \le x_k$ will be

$$x \mapsto \frac{y_k(x - x_{k-1}) + y_{k-1}(x_k - x)}{x_k - x_{k-1}} = mx + b.$$

   □

6. **Solution:**

Equispaced piecewise linear evaluation

```
epwlinear( x, y[], a, b, N ):
[0] If x<a, then return y[0]
[1] If x>b, then return y[N]
[2] Let h=(b-a)/N, let k=floor((x-a)/h), let dx=x-k*h
[3] Return (y[k+1]*dx+y[k]*(h-dx))/h
```

□

7. **Solution:** First note that $y_1/y_2 < 0$ and $y_2/y_1 < 0$ if $y_1 y_2 < 0$. Then

$$x_0 - x_1 = \frac{x_2 y_1 - x_1 y_2 - x_1 y_1 + x_1 y_2}{y_1 - y_2} = \frac{(x_2 - x_1) y_1}{y_1 - y_2} = \frac{x_2 - x_1}{1 - \frac{y_2}{y_1}} > 0,$$

and

$$x_0 - x_2 = \frac{x_2 y_1 - x_1 y_2 - x_2 y_1 + x_2 y_2}{y_1 - y_2} = \frac{y_2 (x_2 - x_1)}{y_1 - y_2} = \frac{x_2 - x_1}{\frac{y_1}{y_2} - 1} < 0.$$

□

8. **Solution:** First note that the formula for x_* is preserved by a change of sign $(y_1, y_2, y_3) \mapsto (-y_1, -y_2, -y_3)$. Thus we may assume without loss of generality that $y_2 > y_1$ and $y_2 > y_3$, so x_* will be a local maximum.

Second, note that the denominators of $x_* - x_1$ and $x_3 - x_*$ will be the same, namely

$$\begin{aligned} D &= x_1(y_3 - y_2) + x_2(y_1 - y_3) + x_3(y_2 - y_1) \\ &= y_1(x_2 - x_3) + y_2(x_3 - x_1) + y_3(x_1 - x_2). \end{aligned}$$

Putting $p = x_2 - x_1 > 0$ and $q = x_3 - x_2 > 0$, so that $p + q = x_3 - x_1 > 0$, we see that

$$D = (p+q)\left(y_2 - \frac{p}{p+q} y_3 - \frac{q}{p+q} y_1\right) > 0,$$

since $\frac{p}{p+q} y_3 + \frac{q}{p+q} y_1$ lies between y_1 and y_3, hence below y_2.

It remains to show that the numerators of $x_* - x_1$ and $x_3 - x_*$ are both positive. But,

$$\begin{aligned} x_* - x_1 &= \frac{1}{2D}\big(- x_1^2(y_3 - y_2) + (x_2^2 - 2x_1x_2)(y_1 - y_3) \\ &\qquad + (x_3^2 - 2x_1x_3)(y_2 - y_1)\big) \\ &= \frac{1}{2D}\big((x_1^2 + x_3^2 - 2x_1x_3)y_2 - (x_2^2 + x_1^2 - 2x_1x_2)y_3 \\ &\qquad - (x_3^2 - 2x_1x_3 - x_2^2 + 2x_1x_2)y_1\big) \\ &= \frac{1}{2D}\big((x_3 - x_1)^2 y_2 - (x_2 - x_1)^2 y_3 \\ &\qquad - \left[(x_3 - x_1)^2 - (x_2 - x_1)^2\right] y_1\big). \end{aligned}$$

Putting $r = (x_2 - x_1)^2 > 0$ and $s = (x_3 - x_1)^2 - (x_2 - x_1)^2 > 0$ gives $r + s = (x_3 - x_1)^2 > 0$, so

$$x_* - x_1 = \frac{1}{2D}(r+s)\left(y_2 - \frac{r}{r+s}y_3 - \frac{s}{r+s}y_1\right) > 0,$$

since $\frac{r}{r+s}y_3 + \frac{s}{r+s}y_1$ lies between y_1 and y_3, hence below y_2.
Likewise,

$$\begin{aligned} x_3 - x_* &= \frac{1}{2D}\big((2x_1x_3 - x_1^2)(y_3 - y_2) + (2x_3x_2 - x_2^2)(y_1 - y_3) \\ &\qquad + x_3^2(y_2 - y_1)\big) \\ &= \frac{1}{2D}\big((x_1^2 + x_3^2 - 2x_1x_3)y_2 - (x_3^2 + x_2^2 - 2x_1x_2)y_1 \\ &\qquad - (x_1^2 - 2x_1x_3 - x_2^2 + 2x_3x_2)y_3\big) \\ &= \frac{1}{2D}\big((x_3 - x_1)^2y_2 - (x_3 - x_2)^2y_1 \\ &\qquad - \left[(x_3 - x_1)^2 - (x_3 - x_2)^2\right]y_3\big). \end{aligned}$$

Putting $u = (x_3 - x_2)^2 > 0$ and $v = (x_3 - x_1)^2 - (x_3 - x_2)^2 > 0$ gives $u + v = (x_3 - x_1)^2 > 0$, so

$$x_3 - x_* = \frac{1}{2D}(u+v)\left(y_2 - \frac{u}{u+v}y_1 - \frac{v}{u+v}y_3\right) > 0,$$

since $\frac{u}{u+v}y_1 + \frac{v}{u+v}y_3$ lies between y_1 and y_3, hence below y_2. □

9. **Solution:** For $M \le N$, define the partial sum

$$(Pu)_{MN} = \sum_{n=M}^{N} \langle \phi_n, u \rangle \, \phi_n,$$

so $(Pu)_{0N} = \sum_{n=0}^{N} \langle \phi_n, u \rangle \, \phi_n$ for any $N \ge 0$. By Lemma 2.9,

$$\|(Pu)_{0N}\|^2 = \sum_{n=0}^{N} |\langle \phi_n, u \rangle|^2,$$

since $\{\phi_1, \phi_1, \ldots, \phi_N\} \subset \mathbf{B}$ is a finite orthonormal set. By Bessel's inequality (see Exercise 14(a), Chapter 2), we have $\sum_{n=0}^{N} |\langle \phi_n, u \rangle|^2 \le \|u\|^2$ for every $N \ge 0$. This can only happen if for every $\epsilon > 0$, there is some integer $T_\epsilon^+ > 0$ such that $\|(Pu)_{MN}\| < \epsilon$ for all $N \ge M \ge T_\epsilon^+$. A similar argument applied to $(Pu)_{M0}$ shows that for each $\epsilon > 0$ there is some integer $T_\epsilon^- < 0$ such that $M \le N \le T_\epsilon^-$ implies $\|(Pu)_{MN}\| < \epsilon$. Choosing $T_\epsilon \stackrel{\text{def}}{=} \max\{-T_\epsilon^-, T_\epsilon^+\}$ satisfies Definition 3 and shows that $Pu \in \overline{\text{span}}\,\mathbf{B}$. □

10. **Solution:** For integers $M < N$, let $f_{MN}(t) \stackrel{\text{def}}{=} \sum_{n=M}^{N} c(n)\phi(t-n) \in \text{span}\{\phi(t-n) : n \in \mathbf{Z}\}$, and note that this partial sum is bounded, continuous, and compactly supported on the interval $[M-1, N+1]$. Thus $f_{MN} \in L^2(\mathbf{R})$, with $\|f_{MN}\|^2 = \int_{M-1}^{N+1} |f_{MN}(t)|^2\, dt$. But f_{MN} is piecewise linear and interpolates the points $\{(M-1,0); (M,c(M)), \ldots, (N,c(N)); (N+1,0)\}$, so we may compute the square integral in pieces. To do this, we first find the formula for the square integral of a linear function g through $(0,y_0)$ and $(1,y_1)$, using a special case of Equation 4.13:

$$\int_0^1 |g(t)|^2\, dt = \int_0^1 |ty_1 + (1-t)y_0|^2\, dt = \frac{1}{6}\Big(|y_0|^2 + |y_1|^2 + |y_0+y_1|^2\Big).$$

We now apply this formula to evaluate the integral of $|f_{MN}(t)|^2$, which is linear between adjacent integers. Accounting for the end terms, we get:

$$\begin{aligned}
\|f_{MN}\|^2 &= \frac{1}{3}|c(M)|^2 + \frac{1}{3}|c(N)|^2 + \\
&\qquad + \frac{1}{6}\sum_{n=M}^{N-1}\Big(|c(n)|^2 + |c(n{+}1)|^2 + |c(n)+c(n{+}1)|^2\Big) \\
&= \frac{1}{3}\sum_{n=M}^{N}|c(n)|^2 + \\
&\qquad + \frac{1}{6}\left(|c(M)|^2 + \sum_{n=M}^{N-1}|c(n)+c(n{+}1)|^2 + |c(N)|^2\right).
\end{aligned}$$

But $|p+q|^2 \le 2|p|^2 + 2|q|^2$ for any $p, q \in \mathbf{C}$, so we may estimate the second term as follows:

$$0 \le \frac{1}{6}\left(|c(M)|^2 + \sum_{n=M}^{N-1}|c(n)+c(n{+}1)|^2 + |c(N)|^2\right) \le \frac{2}{3}\sum_{n=M}^{N}|c(n)|^2.$$

Therefore,

$$\frac{1}{3}\sum_{n=M}^{N}|c(n)|^2 \le \|f_{MN}\|^2 \le \sum_{n=M}^{N}|c(n)|^2. \tag{A.1}$$

If $f \in L^2(\mathbf{R})$, then $\|f_{-NN}\| \to \|f\|$ as $N \to \infty$, so by the left-hand inequality, the partial sums $\sum_{n=-N}^{N}|c(n)|^2$ increase and thus converge to something no bigger than $3\|f\|^2$ as $N \to \infty$. Therefore, c belongs to ℓ^2.

Conversely, if $c \in \ell^2$, then $\sum_{n=-N}^{N}|c(n)|^2$ converges to $\|c\|^2$ as $N \to \infty$, so for every $\epsilon > 0$ there is some $T = T_\epsilon > 0$ such that for every $N \ge M \ge T$,

$$\sum_{n=M}^{N}|c(n)|^2 + \sum_{n=-N}^{-M}|c(n)|^2 < \epsilon.$$

But for all integers $M' < M < N < N'$, we have

$$f_{M'N'}(t) - f_{MN}(t) = \sum_{n=M'}^{M-1} c(n)\phi(t-n) + \sum_{n=N+1}^{N'} c(n)\phi(t-n),$$

so by the right-hand inequality in Equation A.1, for any $N' > N > T_\epsilon$ and $M' < M < -T_\epsilon$, we will have

$$\|f_{M'N'} - f_{MN}\|^2 \leq \sum_{n=M'}^{M-1} |c(n)|^2 + \sum_{n=N}^{N'} |c(n)|^2 < \epsilon.$$

But this means that the partial sums f_{MN} converge as $N \to \infty$ and $M \to -\infty$ to some limit $f \in \overline{\text{span}}\,\{\phi(t-n) : n \in \mathbf{Z}\}$ satisfying $\|f\|^2 \leq \|c\|^2$, so this f belongs to $L^2(\mathbf{R})$. □

11. **Solution:** a. A counter that always fails to count the first item, but never fails to count all subsequent items, will be inaccurate by one count but with zero variance.

b. A perfect counter has zero imprecision and zero inaccuracy, since recounting will always yield the same correct measurement. But its measurements, nonnegative integers, are quantized with interval 1. □

12. **Solution:** a. First note that $\mathbf{1}(Nt-k) = 1$ if and only if $t \in \left[\frac{k}{N}, \frac{k+1}{N}\right]$. Thus $\|f - f_N\|^2 = \int_0^1 |f - f_N|^2\, dt$ breaks up into

$$\sum_{k=0}^{N-1} \int_{\frac{k}{N}}^{\frac{k+1}{N}} |t - \frac{k}{N}|^2\, dt = \sum_{k=0}^{N-1} \int_0^{\frac{1}{N}} |t|^2\, dt = \frac{1}{3N^2}.$$

We have performed the substitution $t \leftarrow t + \frac{k}{N}$ in the k^{th} integral.

b. Since $\|f\|^2 = \int_0^1 |t|^2\, dt = 1/3$, we simply use Equation 4.36 with $f_s = f$ and $f_q = f_N - f$ to compute $SNR(f_N) = 20\log_1 0N$. □

13. **Solution:** a. That integral is the volume under the graph of f, and $f(x,y) = 2$ is nonzero only if (x,y) lies in the subset of Q bounded by the lines $x = 0$, $y = 1$, and $x = y$. This is the upper-triangular half of Q, and has area $1/2$, so the volume $\iint_Q f(x,y)\, dxdy = (2)(1/2) = 1$.

b. Using Equation 4.38, we get $c_y = 2y$ and so $f(x \mid y) = (2y)^{-1} f(x,y)$.

c. Using Equation 4.37, we compute

$$\Pr(X \in [a,b] \mid y) = \begin{cases} 0, & \text{if } 0 \leq y \leq a; \\ (y-a)/y, & \text{if } a < y < b; \\ (b-a)/y, & \text{if } b \leq y \leq 1. \end{cases}$$

d. Using Equation 4.43, and writing $I = [0,1]$, we compute

$$\begin{aligned}
R^2(d,y) &= \int_I \cdots \int_I |d(x_1,\ldots,x_N) - y|^2 \times \\
&\qquad \times f(x_1 \,|\, y) \cdots f(x_N \,|\, y)\, dx_1 \cdots dx_N \\
&= (2y)^{-N} \int_I \cdots \int_I |\max\{x_1,\ldots,x_N\} - y|^2 \times \\
&\qquad \times f(x_1,y) \cdots f(x_N,y)\, dx_1 \cdots dx_N \\
&= 2^N (2y)^{-N} N! \int_0^y dx_1 \int_0^{x_1} dx_2 \cdots \int_0^{x_{N-1}} |x_1 - y|^2\, dx_N \\
&= y^{-N} N! \int_0^y |x_1 - y|^2 \frac{x_1^{N-1}}{(N-1)!}\, dx_1 \\
&= N y^{-N} \int_0^y x_1^{N-1} |x_1 - y|^2\, dx_1 \\
&= N y^{-N} \left[\frac{x_1^N y^2}{N} - 2\frac{x_1^{N+1} y}{N+1} + \frac{x_1^{N+2}}{N+2} \right] \Bigg|_0^y \\
&= y^2 \left[1 - \frac{2N}{N+1} + \frac{N}{N+2} \right] \quad = \quad \frac{2y^2}{(N+1)(N+2)}.
\end{aligned}$$

The factor $N!$ appears on the third line since that is how many permutations there are of the N variables $0 \le x_N < \cdots < x_1 < y$, which is the only ordering that we integrate. Iterating the $N-1$ inner integrals successively produces x_{N-1}, $x_{N-2}^2/2!$, ..., $x_1^{N-1}/(N-1)!$, as in Equation 4.42, and the remaining steps are straightforward.

Substituting $y \leftarrow E(d \,|\, y) = \frac{Ny}{N+1}$ in the above calculation, which produces $\mathrm{Var}(d \,|\, y) \le R^2(d,y)$, gives

$$\mathrm{Var}(d \,|\, y) = \frac{N}{y^N} \int_0^y x_1^{N-1} \left| x_1 - \frac{Ny}{N+1} \right|^2 dx_1 = \frac{y^2}{(N+1)^2(N+2)}.$$

Thus $\sqrt{\mathrm{Var}(d \,|\, y)} = O(1/\sqrt{N^3})$ and $R(d,y) = O(1/\sqrt{N^2})$ as $N \to \infty$, so both decrease faster than the $1/\sqrt{N}$ rate one might expect from the Cramér-Rao lower bound, which of course does not apply. □

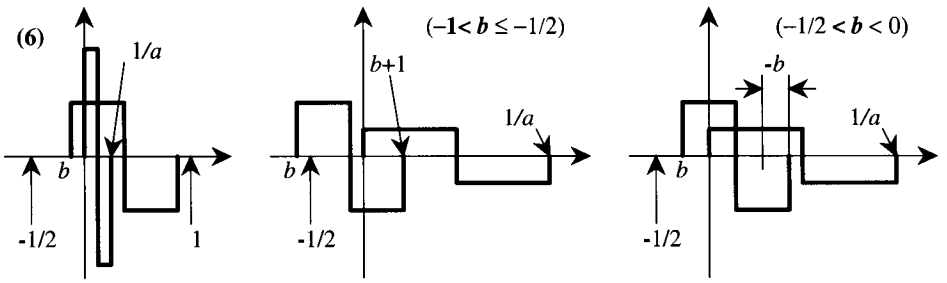


Figure A.1: Graphs of Haar wavelets $w(at)$ and $w(t-b)$ in the configuration of Case 6 and the two subcases of $-1 < b < 0$ and $0 < a < \frac{1}{2(b+1)}$ where their inner product is nonzero.

A.5 ... to Chapter 5 Exercises

1. **Solution:** The graphs are shown in Figure A.1. In Case 6, the integrand of $\int \bar{w}(at)w(t-b)\,dt$ is nonzero only on $[0, \frac{1}{a}]$, where half the time it is $1/\sqrt{a}$ and the other half it is $-1/\sqrt{a}$. To prove Equation 5.3, note that

$$\int \bar{w}(at)w(t-b)\,dt = \int_0^{b+1} (1)(-1)\,dt = -(b+1), \qquad \text{if } -1 < b \leq -\tfrac{1}{2}.$$

Otherwise, if $-\frac{1}{2} < b < 0$, the integral splits into the sum of three pieces:

$$\int_0^{b+\frac{1}{2}} (1)(1)\,dt + \int_{b+\frac{1}{2}}^{2(b+\frac{1}{2})} (1)(-1)\,dt + \int_{2(b+\frac{1}{2})}^{b+1} (1)(-1)\,dt.$$

The first two integrals cancel and the third evaluates to b. Multiplying by $\sqrt{a}$ gives the result. □

2. **Solution:** It is evident that $\rho(a,b) = \rho(a',b')$ if and only if $a = a'$ and $b = b'$, since two matrices are equal if and only if their coefficients are equal. Thus ρ is one-to-one, and it remains to check that ρ preserves the group multiplication. But

$$\begin{aligned} \rho(a',b')\rho(a,b) &= \begin{pmatrix} a' & b' \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} a'a & a'b+b' \\ 0 & 1 \end{pmatrix} \\ &= \rho(a'a, a'b+b') = \rho((a',b')(a,b)). \end{aligned}$$

□

3. **Solution:** Right invariance of the integral requires that the Jacobian of the change of variable $\mathbf{b} \leftarrow \mathbf{b}\mathbf{b}'$ is the constant 1. Since $d\mathbf{b} = k(\mathbf{b})\,dadb =$

$k(a,b)\,da\,db$ for some unknown function k, the right invariance condition can be written:

$$1 = \left|\frac{d(\mathbf{b}\mathbf{b}')}{d\mathbf{b}}\right| = \frac{k(\mathbf{b}\mathbf{b}')}{k(\mathbf{b})}\begin{vmatrix} \frac{da''}{da} & \frac{da''}{db} \\ \frac{db''}{da} & \frac{db''}{db} \end{vmatrix} = \frac{k(\mathbf{b}\mathbf{b}')}{k(\mathbf{b})}\begin{vmatrix} a' & 0 \\ b' & 1 \end{vmatrix} = \frac{k(\mathbf{b}\mathbf{b}')}{k(\mathbf{b})}\,|a'|.$$

Here $\mathbf{b}\mathbf{b}' = (a,b)(a',b') = (a'',b'')$ defines the coordinate functions $a'' = aa'$, $b'' = ab' + b$ from Equation 5.5. Thus $k(\mathbf{b}\mathbf{b}') = \frac{1}{|a'|}k(\mathbf{b})$, so putting $\mathbf{b} = (1,0)$ to force $\mathbf{b}\mathbf{b}' = \mathbf{b}'$ yields $k(a,b) = |a|^{-1}k(1,0)$.

But $a > 0$, so $|a| = a$. Choosing $k(1,0) = 1$ gives the *normalized right invariant integral* of f over **Aff**:

$$\int_{\mathbf{Aff}} f(\mathbf{b})\,d\mathbf{b} \stackrel{\text{def}}{=} \int_{b=-\infty}^{\infty}\int_{a=0}^{\infty} f(a,b)\,\frac{da\,db}{a}.$$

Note that this differs from the left-invariant integral on **Aff**. □

4. **Solution:** Translating Equation 5.21 by $t \leftarrow t - k$, write

$$\sum_{j=1}^{\infty} 2^{-j/2} Wu(2^j, k) = \lim_{J\to\infty} \sum_{j=1}^{J} 2^{-j/2} Wu(2^j, k) = \lim_{J\to\infty} \left\langle \phi_k^J, u \right\rangle,$$

where $\phi_k^J(t) = \phi^J(t-k) = \sum_{j=1}^{J} \frac{1}{2^j} w\left(\frac{t-k}{2^j}\right)$. Equation 5.4 then implies:

$$\phi_k^J(t) = \begin{cases} 0, & \text{if } t < k \text{ or } t \geq 2^J + k; \\ 1 - 2^J, & \text{if } k \leq t \leq k+1; \\ -2^{-J}, & \text{if } k+1 \leq t \leq k + 2^J. \end{cases}$$

Finally,

$$\begin{aligned} \sum_{j=1}^{\infty} 2^{-j/2} Wu(2^j, k) &= \lim_{J\to\infty} \left[(1-2^J) \int_k^{k+1} u(t)\,dt - 2^{-J} \int_{k+1}^{k+2^J} u(t)\,dt \right] \\ &= \int_k^{k+1} u(t)\,dt \quad = u(k). \end{aligned}$$

□

5. **Solution:** Function ϕ is differentiable by calculus techniques everywhere except possibly at the origin, where it must be shown that the limits of ϕ and all its derivatives are zero. But $\lim_{\xi\to 0} \phi(\xi) = 0$ because $-(\log|\xi|)^2 \to -\infty$ as $\xi \to 0$. For the derivatives, it is enough to prove that at each $\xi \neq 0$,

$$\phi^{(d)}(\xi) = \frac{P(\log|\xi|)}{\xi^d}\, e^{-(\log|\xi|)^2}, \tag{A.2}$$

where P is some polynomial. But this can be established for every $d > 0$ by induction.

Starting with $d = 1$, it is immediate that $\phi'(\xi) = \frac{-2\log|\xi|}{\xi} e^{-(\log|\xi|)^2}$ at each $\xi \neq 0$.

Next, assuming that $\phi^{(d)}$ satisfies Equation A.2, calculate

$$\begin{aligned}\phi^{(d+1)}(\xi) &= \frac{d}{d\xi}\phi^{(d)}(\xi) \\ &= \frac{P'(\log|\xi|) - 2\log|\xi|\ P(\log|\xi|) - dP(\log|\xi|)}{\xi^{d+1}} e^{-(\log|\xi|)^2},\end{aligned}$$

which is of the desired form with a new polynomial $Q(z) = -2zP(z) + P'(z) - dP(z)$. This completes the induction step.

To complete the solution, note that

$$\lim_{\xi\to 0} \phi^{(d)}(\xi) = \lim_{\xi\to 0} \frac{P(\log|\xi|)}{\xi^d} e^{-(\log|\xi|)^2} = 0,$$

by L'Hôpital's rule. □

6. **Solution:** For any fixed d and any $n \geq 0$, $\phi^{(n)}(\xi) < 1/|\xi|^d$ as $\xi \to -\infty$, since $\phi^{(n)}(\xi) = 0$ for all $\xi < 0$. To show that for some positive constant C, $|\xi|^d\phi^{(n)}(\xi) \leq C$ as $\xi \to +\infty$, note that $|\xi|^d\phi^{(n)}(\xi) = \xi^d \frac{P(\log\xi)}{\xi^n} e^{-(\log\xi)^2} = \xi^{d-n-\log\xi} P(\log\xi)$, where P is some polynomial depending on n. But this expression will tend to zero as $\xi \to \infty$, since the exponent $d - n - \log\xi$ will become more and more negative as $\xi \to \infty$, eventually cancelling the fixed-degree growth from $P(\log\xi)$. □

7. **Solution:** One solution is to use Plancherel's formula to write:

$$Wu(a,b) = \int_{-\infty}^{\infty} e^{-2\pi ib\xi}\phi(a\xi)\mathcal{F}u(\xi)\,d\xi.$$

Again by Plancherel's theorem, $\mathcal{F}u(\xi)$ is square integrable. Let the other factor, $e^{-2\pi ib\xi}\phi(a\xi)$, be called $\Phi_{ab}(\xi)$. Solution 5 implies that $\Phi_{ab}(\xi)$ is continuous in ξ. For any $a \neq 0$, Solution 6 implies that it decreases rapidly enough as $\xi \to \pm\infty$ to be absolutely integrable: being $O(|\xi|^{-2})$ is sufficient. Thus, it is square-integrable, so $\Phi_{ab}(\xi)\mathcal{F}u(\xi)$ is integrable, and the integral converges absolutely. Next, note that:

$$\frac{\partial}{\partial a}\Phi_{ab}(\xi) = \xi e^{-2\pi ib\xi}\phi'(a\xi); \qquad \frac{\partial}{\partial b}\Phi_{ab}(\xi) = -2\pi i\xi e^{-2\pi ib\xi}\phi(a\xi);$$

$$\frac{\partial^2}{\partial a^2}\Phi_{ab}(\xi) = \xi^2 e^{-2\pi ib\xi}\phi''(a\xi); \qquad \frac{\partial^2}{\partial b^2}\Phi_{ab}(\xi) = -4\pi^2\xi^2 e^{-2\pi ib\xi}\phi(a\xi);$$

$$\frac{\partial^2}{\partial a\,\partial b}\Phi_{ab}(\xi) = \frac{\partial^2}{\partial b\,\partial a}\Phi_{ab}(\xi) = -2\pi i\xi^2 e^{-2\pi ib\xi}\phi'(a\xi).$$

By Solutions 5 and 6, all of these are continuous in ξ and, for $a \neq 0$, decrease fast enough to be absolutely integrable and square-integrable with respect to

ξ. Thus, it is legal to differentiate once or twice with respect to a or b under the integral sign:

$$\begin{aligned}\frac{\partial}{\partial a}Wu(a,b) &= \int_{-\infty}^{\infty} \xi e^{-2\pi i b\xi}\phi'(a\xi)\mathcal{F}u(\xi)\,d\xi;\\ \frac{\partial}{\partial b}Wu(a,b) &= -2\pi i\int_{-\infty}^{\infty} \xi e^{-2\pi i b\xi}\phi(a\xi)\mathcal{F}u(\xi)\,d\xi.\end{aligned}$$

It remains to show that these derivatives are continuous functions of a, b away from the line $a = 0$. But in both cases, this follows from the observation that the integrands are continuous functions of a, b. □

8. **Solution:** Since $w(x) = \widehat{\chi_{[-1/2,1/2]}}(x)$, Plancherel's theorem implies $\|w\| = \|\chi_{[-1/2,1/2]}\| = 1$. □

9. **Solution:** By the previous solution and by combining integrals, calculate that $\mathcal{F}w = \chi_{[-1,-1/2]\cup[1/2,1]}$. Thus,

$$c_w = \int \chi_{[-1/,-1/2]\cup[1/2,1]}(\xi)\frac{d\xi}{|\xi|} = 2\int_{\frac{1}{2}}^{1}\frac{d\xi}{\xi} = 2\log 2 < \infty,$$

which in particular means that w is admissible. □

10. **Solution:** The Fourier integral transform of w is

$$\int_{-\infty}^{\infty} e^{-2\pi i x\xi}w(x)\,dx.$$

Since $w(x) = 1$ if $0 < x < \frac{1}{2}$ and $w(x) = -1$ if $\frac{1}{2} < x < 1$, that simplifies to

$$\int_0^{\frac{1}{2}} e^{-2\pi i x\xi} - \int_{\frac{1}{2}}^{1} e^{-2\pi i x\xi} = \frac{\left(e^{-\pi i\xi}-1\right)^2}{2\pi i\xi}.$$

□

11. **Solution:** It is necessary to show that $\langle\phi_j,\phi_k\rangle = \delta(j-k)$. But Plancherel's theorem allows writing

$$\langle\phi_j,\phi_k\rangle = \left\langle\widehat{\phi_j},\widehat{\phi_k}\right\rangle = \int_{-1/2}^{1/2} e^{2\pi i(k-j)\xi}\,d\xi = \delta(j-k),$$

since $\widehat{\phi_k}(\xi) = e^{2\pi i k\xi}\widehat{\mathrm{sinc}}\,(\xi) = e^{2\pi i k\xi}\chi_{[-1/2,1/2]}(\xi)$. □

12. **Solution:** Show that $\sum_k g(2k) = -\sum_k g(2k+1) = \frac{1}{\sqrt{2}}$:

$$\begin{aligned}\sum_k g(2k) &= \sum_k (-1)^{2k}h(2M-1-2k) = \sum_k h(2M-1-2k)\\ &= \sum_k h(2(M-k)-1) = \sum_l h(2l+1) = \frac{1}{\sqrt{2}},\end{aligned}$$

after the substitution $l \leftarrow M - k$. A similar change of the index gives:

$$\begin{aligned}\sum_k g(2k+1) &= \sum_k (-1)^{2k+1} h(2M-1-(2k+1)) \\ &= -\sum_k h(2(M-k-1)) = -\sum_l h(2l) = -\frac{1}{\sqrt{2}}.\end{aligned}$$

□

13. **Solution:** First note that $1 + c^2 = 8 - 4\sqrt{3} = 4c$, so $c/(1+c^2) = \frac{1}{4}$. Thus

$$h(2) = \frac{c(c+1)}{\sqrt{2}(1+c^2)} = \frac{c+1}{4\sqrt{2}} = \frac{(2-\sqrt{3})+1}{4\sqrt{2}} = \frac{3-\sqrt{3}}{4\sqrt{2}}.$$

Likewise,

$$h(3) = \frac{c(c-1)}{\sqrt{2}(1+c^2)} = \frac{c-1}{4\sqrt{2}} = \frac{1-\sqrt{3}}{4\sqrt{2}}.$$

Finally,

$$h(0) = \frac{1}{\sqrt{2}} - h(2) = \frac{1+\sqrt{3}}{4\sqrt{2}}; \qquad h(1) = \frac{1}{\sqrt{2}} - h(3) = \frac{3+\sqrt{3}}{4\sqrt{2}}.$$

□

14. **Solution:** We begin by substituting $t \leftarrow 2^{-j-1}t$ and then multiplying by $2^{-(j+1)/2}$ on both sides of Equation 5.33:

$$\begin{aligned}2^{-(j+1)/2}\phi(2^{-j-1}t - n) &= \sum_k h(k) 2^{-j/2} \phi(2^{-j}t - 2n - k) \\ &= \sum_{k'} h(k' - 2n) 2^{-j/2} \phi(2^{-j}t - k').\end{aligned}$$

Taking inner products on both sides with u gives the first result.

Similarly, we may substitute $t \leftarrow 2^{-j-1}t - n$ and then multiply by $2^{-(j+1)/2}$ in Equation 5.50 to get

$$\begin{aligned}2^{-(j+1)/2}\psi(2^{-j-1}t - n) &= \sum_k g(k) 2^{-j/2} \phi(2^{-j}t - 2n - k) \\ &= \sum_{k'} g(k' - 2n) 2^{-j/2} \phi(2^{-j}t - k').\end{aligned}$$

Taking inner products on both sides with u gives the second result.

For the third result, we take inner products with u on both sides of Equation 5.55. □

15. **Solution:** First check the odd and even cases to show that for any integers p, q,

$$\left\lceil \frac{p}{2} \right\rceil \leq \frac{p+1}{2}; \qquad \left\lfloor \frac{q}{2} \right\rfloor \geq \frac{q-1}{2}.$$

a. Substitute $p \leftarrow x - b$ and $q \leftarrow y - a$ to get

$$\begin{aligned} 2\left\lceil \frac{x-b}{2} \right\rceil + a &\leq 2\left(\frac{x-b+1}{2}\right) + a = x - (b-a-1); \\ 2\left\lfloor \frac{y-a}{2} \right\rfloor + b &\geq 2\left(\frac{y-a-1}{2}\right) + b = y + (b-a-1). \end{aligned}$$

Thus $\left[2\left\lceil \frac{x-b}{2} \right\rceil + a, 2\left\lfloor \frac{y-a}{2} \right\rfloor + b\right] \supseteq [x-(b-a-1), y+(b-a-1)]$, with equality if and only if $b - a = 1$.

b. Substitute $p \leftarrow x - b$ and $q \leftarrow y - a$ to get

$$1 + \left\lfloor \frac{y-a}{2} \right\rfloor - \left\lceil \frac{x-b}{2} \right\rceil \geq 1 + \frac{y-a-1}{2} - \frac{x-b+1}{2}.$$

Then substitute $p \leftarrow x - d$ and $q \leftarrow y - c$ to get

$$1 + \left\lfloor \frac{y-c}{2} \right\rfloor - \left\lceil \frac{x-d}{2} \right\rceil \geq 1 + \frac{y-c-1}{2} - \frac{x-d+1}{2}.$$

Adding these inequalities and simplifying the smaller right-hand side to $1 + y - x + \frac{b-a-1}{2} + \frac{d-c-1}{2}$ finishes the proof. □

16. **Solution:** First note that if $P = 2P'$ is even, then for each $k \in \mathbf{Z}$ there are unique $j \in \mathbf{Z}$ and $k' \in \{0, 1, \ldots, P'-1\}$ such that $2k = 2(k' + jP')$ and $2k + 1 = 2(k' + jP') + 1$. Thus, for any finitely-supported sequence f with P-periodization f_P, we can sum

$$\sum_{k'=0}^{P'-1} f_P(2k') = \sum_{k'=0}^{P'-1} \sum_j f(2k' + jP) = \sum_{k'=0}^{P'-1} \sum_j f(2(k' + jP')) = \sum_k f(2k),$$

and likewise,

$$\begin{aligned} \sum_{k'=0}^{P'-1} f_P(2k'+1) &= \sum_{k'=0}^{P'-1} \sum_j f(2k' + jP + 1) \\ &= \sum_{k'=0}^{P'-1} \sum_j f(2(k' + jP') + 1) = \sum_k f(2k+1). \end{aligned}$$

Substituting normalized h or g for f shows that normalization holds for h_P and g_P.

Second, note that if e and f are finitely-supported sequences with respective P-periodizations e_P and f_P, then

$$\begin{aligned}\sum_{k'=0}^{P-1} e_P(k'+2n)f_P(k'+2m) &= \\ &= \sum_{k'=0}^{P-1}\sum_j\sum_i e(k'+2n+jP)f(k'+2m+iP) \\ &= \sum_{k'=0}^{P-1}\sum_j\sum_l e(k'+jP+2n)f(k'+jP+2(m+lP')),\end{aligned}$$

after substituting $i \leftarrow l+j$. The sums over $0 \le k' < P$ and $j \in \mathbf{Z}$ combine into a sum over all integers $k \in \mathbf{Z}$, and the l and k sums may be interchanged, giving

$$\sum_{k'=0}^{P-1} e_P(k'+2n)f_P(k'+2m) = \sum_l\sum_k e(k+2n)f(k+2(m+lP')).$$

If $e = f = h$ and h is self-orthonormal, then the inner sum over k is $\delta(n-(m+lP'))$. Thus the outer sum over l is $\delta_{P'}(n-m)$, which is 1 if and only if $n \equiv m \pmod{P'}$; otherwise it is zero. The same holds if $e = f = g$ is self-orthonormal. If $e = h$ and $f = g$ are independent, the inner sum over k is always zero, so the total is zero. This establishes periodic independence and self-orthonormality.

Finally,

$$\begin{aligned}\sum_{k'=0}^{P'-1} f_P(2k'+n)f_P(2k'+m) &= \\ &= \sum_{k'=0}^{P'-1}\sum_j\sum_i f(2k'+n+jP)f(2k'+m+iP) \\ &= \sum_{k'=0}^{P'-1}\sum_j\sum_i f(2(k'+jP')+n)f(2(k'+jP')+m+(i-j)P),\end{aligned}$$

so substituting $i \leftarrow l+j$ and $k' \leftarrow k-jP'$ and combining the k' and j summations into one makes this

$$\sum_k\sum_l f(2k+n)f(2k+m+lP).$$

The l and k sums may be interchanged. The cases $f \leftarrow h$ and $f \leftarrow g$ give

$$\sum_{k'=0}^{P'-1} h_P(2k+n)h_P(2k+m) = \sum_l\sum_k h(2k+n)h(2k+m+lP);$$

$$\sum_{k'=0}^{P'-1} g_P(2k+n)g_P(2k+m) \quad = \quad \sum_l \sum_k g(2k+n)g(2k+m+lP).$$

If h and g satisfy the completeness condition, adding these together gives $\sum_l \delta(n-(m+lP)) = \delta_P(n-m)$, proving periodic completeness. □

17. **Solution:** The following is a Standard C implementation. We begin by implementing the inverse filter transform:

Contents of `ipcqfilt.c`

```
int mod(int x, int M) {   /* x%M for M>1 and any x */
  if(x<0) x-= x*M; /* x-x*modulus>0 equals x mod M */
  return x%M;
}
void ipcqfilter(float out[], const float in[], int q) {
  int n2, k2;
  for(k2=0; k2<q; k2++) {
    out[2*k2]=out[2*k2+1]=0;
    for(n2=0;n2<L/2;n2++) {
      out[2*k2] += h[2*n2]*in[mod(k2-n2, q)];
      out[2*k2] += g[2*n2]*in[mod(k2-n2, q) + q];
      out[2*k2+1] += h[2*n2+1]*in[mod(k2-n2, q)];
      out[2*k2+1] += g[2*n2+1]*in[mod(k2-n2, q) + q];
    }
  }
}
```

Notice that these functions will work with filters of any even length L.

Next, we implement the inverse to Mallat's periodic discrete wavelet transform on $N = 2^J K$ samples, generalizing `ipdwt0()`:

Reconstruction from Mallat's Periodic Wavelet Expansion

```
ipdwt( u[], N, J, h[], g[], L ):
[0] If J>0, then do [1-] to [4]
[1]    Compute ipdwt(u[], N/2, J-1, h[], g[], L)
[2]    Allocate temp[0]=0,...,temp[N-1]=0
[3]    Compute ipcqfilter(temp[], u[], N/2, h[], g[], L)
[4]    For i=0 to N-1, let u[i] = temp[i]
```

For practical reasons, we should place the allocation and deallocation of `temp[]` as close as possible to the filter transform. This frees unneeded memory for the recursive function call. In Standard C, this becomes:

Contents of `ipdwt.c`

```
#include <assert.h>
#include <stdlib.h>
void ipdwt(float u[], int N, int J) {
  if(J>0) {
    float *temp; int i;
    ipdwt(u, N/2, J-1);
    temp=(float *)calloc(N,sizeof(float)); assert(temp);
    ipcqfilter(temp, u, N/2);
    for(i=0;i<N;i++) u[i]=temp[i];
    free(temp);
  }
}
```

Finally, we generate the samples for a graph of the Daubechies 4 wavelet and scaling function. We start with an array containing a single nonzero wavelet coefficient at what would be level 5 of a 128-point periodic discrete wavelet transform, and invert to get the one-wavelet signal that would have produced it:

Standard C Program: Graph Daubechies 4 Functions

```
#include <stdio.h>
#define L 4
static const float h[L] =
 {  0.48296291314453416,  0.83651630373780794,
    0.22414386804201339, -0.12940952255126037};
static const float g[L] =
 { -0.12940952255126037, -0.22414386804201339,
    0.83651630373780794, -0.48296291314453416};
#include "ipcqfilt.c"
#include "ipdwt.c"
int main(void) {
  float scaling[128]={0}, wavelet[128]={0};  int i;
  scaling[1]=1.0;  wavelet[5]=1.0;
  ipdwt( scaling, 128, 5 );
  puts("# 128 point Daubechies 4 scaling, level 5:");
  for(i=0; i<128; i++) printf("%10.6f\n", scaling[i]);
  ipdwt( wavelet, 128, 5 );
  puts("# 128 point Daubechies 4 wavelet, level 5:");
  for(i=0; i<128; i++) printf("%10.6f\n", wavelet[i]);
  return 0;
}
```

The indices 1 and 5 are chosen so that the functions, of scale $2^{5-7} = \frac{1}{4}$ relative to the width of the plot, are offset by 1 from the left edge and thus

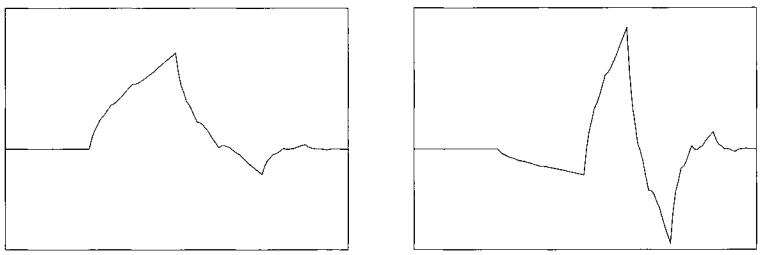


Figure A.2: Graphs of Daubechies 4 wavelets. Left: scaling function ϕ. Right: mother function ψ.

approximately centered. The results are plotted in Figure A.2. □

18. **Solution:** a. After steps 1 and 2, $u(2n+1)$ contains $u(2n+1) - u(2n)$ and $u(2n)$ contains $\frac{1}{2}[u(2n+1) + u(2n)]$. After steps 3 and 4, those values are scaled so that $u(2n+1)$ contains $[u(2n+1) - u(2n)]/\sqrt{2}$ and $u(2n)$ contains $[u(2n+1) + u(2n)]/\sqrt{2}$. This shows that `haarlift()` implements a combined filter transform with the Haar filters of Equation 5.46.

b. After steps 1' and 2', $u(2n)$ contains $u(2n)/sqrt2$ and $u(2n+1)$ contains $\sqrt{2}\,u(2n+1)$. After steps 3' and 4', those values are combined so that $u(2n)$ contains $[u(2n) - u(2n+1)]/\sqrt{2}$ and $u(2n+1)$ contains $[u(2n) + u(2n+1)]/\sqrt{2}$. This shows that `ihaarlift()` implements an adjoint filter transform with the Haar filters of Equation 5.46.

c. Write $v(2n) \stackrel{\text{def}}{=} [u(2n+1) + u(2n)]/\sqrt{2}$ and $v(2n+1) \stackrel{\text{def}}{=} [u(2n+1) - u(2n)]/\sqrt{2}$ for $n = 0, 1, \ldots, q-1$. Then v is a copy of u after the (a) substitutions. Applying the (b) substitutions leaves $[v(2n) - v(2n+1)]/\sqrt{2} = u(2n)$ at index $2n$, and $[v(2n) + v(2n+1)]/\sqrt{2} = u(2n+1)$ at index $2n+1$.

d. We use induction on J to prove that for any $dq \in \mathbf{Z}^+$, the function `ldht0(u[],J,dq)` is inverted by `ildht0(u[],J,dq)`.

Case $J = 0$ holds since both `ldht0(u[],0,dq)` and `ldht0(u[],0,dq)` are the identity on one-point signals u.

For $J > 0$, assume that `ildht0(u[],J-1,dq)` recovers the array u after it is transformed by `ildht0(u[],J-1,dq)`, for all $dq \in \mathbf{Z}^+$. Now consider `ldht0(u[],J,dq)`. This consists of `haarlift()` applied to the array elements $u(0), u(dq), \ldots, u([2^J - 1]dq)$, followed by `ldht0(u[],J-1,2*dq)` which acts just on the even elements $u(0), u(2dq), \ldots, u([2^J - 2]dq)$. By the inductive hypothesis, that is inverted by `ildht0(u[],J-1,2*dq)`, which inverts the Haar-transformed even elements $u(0), u(2dq), \ldots, u([2^J - 2]dq)$, followed by `ihaarlift()` applied to $u(0), u(dq), \ldots, u([2^J - 1]dq)$. □

19. **Solution:** The filter transform `wsl97filter()` is already implemented, so we simply use it recursively, following `wsl42dwt()`:

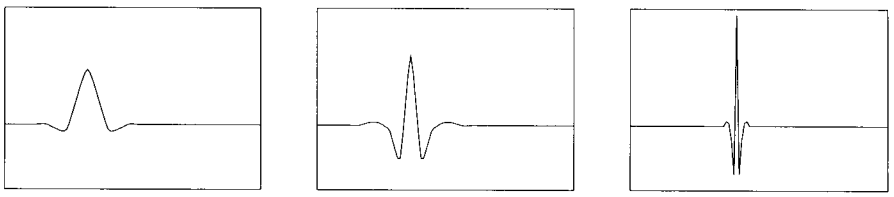


Figure A.3: Graphs of 9,7-biorthogonal wavelets on 100 points. Left: index 32, level 3. Middle: index 36, level 4. Right: index 41, level 5.

WS Lifting: 9,7-Biorthogonal Discrete Wavelet Transform

```
wsl97dwt( u[], N, dq, J ):
[0] If J>0, then do [1] to [2]
[1]    Compute wsl97filter( u[], N, dq )
[2]    Compute wsl97dwt( u[], N, 2*dq, J-1 )
```

The inverse filter transform requires one normalization, two updatings, and two predictions, with inverted coefficients:

WS Lifting: Inverse 9,7-Biorthogonal Filter Transform

```
wsl97ifilter( u[], N, dq ):
[0] Compute lnormalize( u[], N, dq, 1/zeta97 )
[1] Compute wslupdate( u[], N, dq, -delta97 )
[2] Compute wslpredict( u[], N, dq, -gamma97 )
[3] Compute wslupdate( u[], N, dq, -beta97 )
[4] Compute wslpredict( u[], N, dq, -alpha97 )
```

Reconstruction from the output coefficients is accomplished by the inverse:

WS Lifting: 9,7-Biorthogonal Inverse Wavelet Transform

```
wsl97idwt( u[], N, dq, J ):
[0] If J>0, then do [1] to [2]
[1]    Compute wsl97idwt( u[], N, 2*dq, J-1 )
[2]    Compute wsl97ifilter( u[], N, dq )
```

To plot the requested wavelets, we create three arrays u_0, u_1, u_2 of 100 locations each. We put zeroes everywhere except for $u_0(16) = 1$, $u_1(16) = 1$, and $u_2(31) = 1$, then call `wsl97idwt(u0,100,1,3)`, `wsl97idwt(u1,100,1,4)`, and `wsl97idwt(u2,100,1,5)`. The piecewise linear functions through the three resulting sequences $\{(k, u_i(k)) : 0 \leq k < 100\}$, $i = 1, 2, 3$, are plotted in Figure A.3. □

20. **Solution:** First implement half-sample symmetric prediction and updating, by modifying `wslpredict()` and `wslupdate()`:

HS Lifting: Prediction Step

```
hslpredict( u[], N, dq, coeff ):
[0] Let i = dq
[1] While i<N, do [2] to [3]
[2]    Accumulate  u[i] += coeff*u[i-dq]
[3]    Increment  i += 2*dq
```

HS Lifting: Updating Step

```
hslupdate( u[], N, dq, coeff ):
[0] Let i = 0
[1] While i+dq<N, do [2] to [3]
[2]    Accumulate  u[i] += coeff*u[i+dq]
[3]    Increment  i += 2*dq
```

To implement the Haar filter transform by lifting, we predict with $\alpha = -1$, update with $\beta = \frac{1}{2}$, and normalize with $\zeta = \sqrt{2}$. When N/dq is odd, the last coordinate should be left alone, rather than divided by `coeff`, if we want the transform to be orthogonal:

Orthogonal Lifting: Normalization Step

```
onormalize( u[], N, dq, coeff ):
[0] Let i = 0
[1] While i+dq<N, do [2] to [4]
[2]    Replace  u[i] *= coeff
[3]    Replace  u[i+dq] /= coeff
[4]    Increment  i += 2*dq
```

The inverse has the same operations in reverse order, with inverse coefficients:

HS Lifting: Haar Filter Transform

```
hslhaarfilter( u[], N, dq ):
[0] Compute hslpredict( u[], N, dq, alpha )
[1] Compute hslupdate( u[], N, dq, beta )
[2] Compute onormalize( u[], N, dq, zeta )
```

HS Lifting: Inverse Haar Filter Transform

```
hslihaarfilter( u[], N, dq ):
[0] Compute onormalize( u[], N, dq, 1/zeta )
[1] Compute hslupdate( u[], N, dq, -beta )
[2] Compute hslpredict( u[], N, dq, -alpha )
```

Finally, inspired by `wsl42dwt()` and `wsl42idwt()`, we employ these filter transforms recursively to get the discrete wavelet transform and its inverse:

HS Lifting: Discrete Haar Transform

```
hsldht( u[], N, dq, J ):
[0] If J>0, then do [1] to [2]
[1]    Compute hslhaarfilter( u[], N, dq )
[2]    Compute hsldht( u[], N, 2*dq, J-1 )
```

HS Symmetric Lifting: Inverse Discrete Haar Transform

```
hslidht( u[], N, dq, J ):
[0] If J>0, then do [1] to [2]
[1]    Compute hslidht( u[], N, 2*dq, J-1 )
[2]    Compute hslihaarfilter( u[], N, dq )
```

To check the sum-of-squares preserving property, we fix a length N, generate an array u of N random numbers with $\|u\| = 1$, save a copy in array v, apply `hsldht(u,N,1,J)`, and check whether $|\|u\| - \|v\|\|$ is less than a few truncation errors ϵ_f. Here $J \stackrel{\text{def}}{=} \lceil \log_2 N \rceil$ is the deepest level of decomposition possible with N elements.

To check invertibility, we then apply `hslidht(u,N,1,J)` and see whether $\|u - v\|$ is less than a few truncation errors. Note that both `lnormalize()` and `onormalize()` will yield perfect reconstruction, but `onormalize()` will yield an orthogonal Haar transformation. □

21. **Solution:** First note that e_0, e_1, e_2, e_3 are step functions, so the inner product

$$\langle e_i, e_j \rangle = \int_{\mathbf{R}^2} e_i(x,y) e_j(x,y)\, dxdy, quadi, j \in \{0,1,2,3\},$$

can be evaluated by a finite Riemann sum. We can also iterate the integration and write

$$\|e_i\|^2 = \int_{\mathbf{R}^2} e_i^2(x,y)\, dxdy = \left(\int_{\mathbf{R}} a^2(x)\, dx\right)\left(\int_{\mathbf{R}} b^2(y)\, dy\right) = \|a\|^2\|b\|^2,$$

where a and b are either $\mathbf{1}$ or w, depending on i. But then $\|a\| = \|b\| = 1$, proving that $\|e_i\| = 1$ for $i = 0,1,2,3$. Likewise, if $i \neq j$ and $e_i(x,y) = a_i(x)b_i(y)$ and $e_j(x,y) = a_j(x)b_j(y)$ with $a_i, b_i, a_j, b_j \in \{w, \mathbf{1}\}$, then

$$\begin{aligned}\int_{\mathbf{R}^2} e_i(x,y)e_j(x,y)\, dxdy &= \left(\int_{\mathbf{R}} a_i(x)a_j(x)\, dx\right)\left(\int_{\mathbf{R}} b_i(y)b_j(y)\, dy\right) \\ &= \langle a_i, a_j\rangle \langle b_i, b_j\rangle = 0,\end{aligned}$$

since at least one of $\langle a_i, a_j\rangle = 0$ or $\langle b_i, b_j\rangle = 0$. □

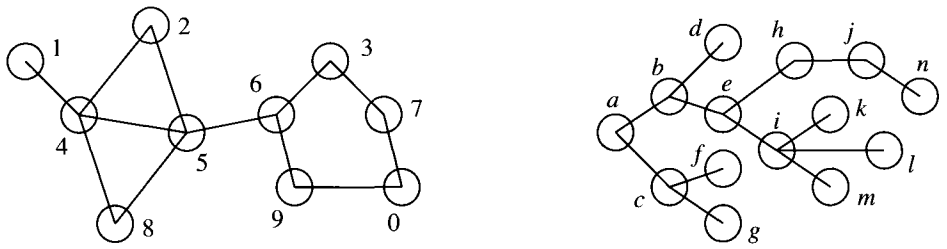


Figure A.4: Two example graphs with labeled vertex sets $V_1 = \{0, 1, \ldots, 9\}$ and $V_2 = \{a, b, \ldots, n\}$.

A.6 ... to Chapter 6 Exercises

1. **Solution:** Figure A.4 gives the labeling, and the edge lists are respectively

$$\begin{aligned} E_1 \;=\; & \{\{4,1\},\{4,2\},\{4,5\},\{4,8\}, \\ & \{5,2\},\{5,6\},\{5,8\},\{6,3\}, \\ & \{6,9\},\{9,0\},\{3,7\},\{7,0\}\}, \end{aligned}$$

   for the left graph, and

$$\begin{aligned} E_2 \;=\; & \{\{a,b\},\{a,c\},\{b,d\},\{b,e\}, \\ & \{c,f\},\{c,g\},\{e,h\},\{e,i\}, \\ & \{h,j\},\{i,k\},\{i,l\},\{i,m\},\{j,n\}\}, \end{aligned}$$

   for the right tree. If we designate a as the root, then generation 0 is $\{a\}$, generation 1 is $\{b, c\}$, generation 2 is $\{d, e, f, g\}$, generation 3 is $\{h, i\}$, generation 4 is $\{j, k, l, m\}$, and generation 5 is $\{n\}$. □

2. **Solution:** None exists, since prefix codes are uniquely decipherable codes and $2^{-1}+2^{-2}+2^{-2}+2^{-3} = 9/8 > 1$ fails the necessary condition for existence stated in Lemma 6.1. □

3. **Solution:** The receiver must keep track of the decoded message and alter the decoding every time a 'c' is encountered:

Dynamic Decoding Example

```
dynamicdecoding( bit[], L ):
[ 0] Initialize n=0 and k=1
[ 1] While k<=L, do [2] to [20]
[ 2]    If n==0, then do [3] to [9]
[ 3]       If bit[k]==1, then do [4] to [5]
[ 4]          Let OUT='a'
[ 5]          Increment k += 1
[ 6]       Else do [7] to [9]
[ 7]          If bit[k+1]==0, then let OUT = 'b'
[ 8]          Else let OUT = 'c'
[ 9]          Increment k += 2
[10]    Else do [11] to [17]
[11]       If bit[k]==0, then do [12] to [13]
[12]          Let OUT = 'a'
[13]          Increment k += 1
[14]       Else do [15] to [17]
[15]          If bit[k+1]==1, then let OUT='b'
[16]          Else let OUT = 'c'
[17]          Increment k += 2
[18]    If OUT=='c', then toggle n = 1-n
[19]    Print OUT
[20]    Go to [1]
```

□

4. **Solution:** First note that

$$\begin{aligned}\sum_{x\in A^s} p(x_1x_2\cdots x_s) &= \sum_{x\in A^s} p(x_1)p(x_2)\cdots p(x_s)\\ &= \sum_{x_1\in A}\sum_{x_2\in A}\cdots\sum_{x_s\in A} p(x_1)p(x_2)\cdots p(x_s)\\ &= \left(\sum_{x_1\in A} p(x_1)\right)\cdots\left(\sum_{x_s\in A} p(x_s)\right) = 1.\end{aligned}$$

Thus p^s gives the occurrence probabilities on A^s. We compute its entropy:

$$\begin{aligned}H(p^s) &= \sum_{x\in A^s} p(x_1x_2\cdots x_s)\log_2\frac{1}{p(x_1x_2\cdots x_s)}\\ &= \sum_{x\in A^s} p(x_1)p(x_2)\cdots p(x_s)\left(\sum_{i=1}^{s}\log_2\frac{1}{p(x_i)}\right)\\ &= \sum_{i=1}^{s}\left[\left(\sum_{x_i\in A} p(x_i)\log_2\frac{1}{p(x_i)}\right)\prod_{j\neq i}\left(\sum_{x_j\in A} p(x_j)\right)\right],\end{aligned}$$

where we have interchanged the order of summation and then extracted one factor $p(x_i)$ from the i^{th} new summand to combine with its corresponding logarithm. But each x_i is an independent random variable with occurrence probabilities p, so for all $k = 1, \ldots, s$,

$$\sum_{x_k \in A} p(x_k) = 1 \qquad \text{and} \qquad \sum_{x_k \in A} p(x_k) \log_2 \frac{1}{p(x_k)} = H(p),$$

independent of k. We conclude that $H(p^s) = \sum_{i=1}^{s} H(p) = sH(p)$. □

5. **Solution:** Let N_d be the number of binary trees of depth d or less. Then $N_0 = 1$, since the only depth=0 binary tree is the one consisting of just the root. Also $N_1 = 4$, with the only possibilities being root, root+son, root+daughter, or root+son+daughter.

   Now observe that any tree of depth at most $d+1$ consists of a root with a possibly empty left subtree of depth at most d, and a possibly empty subtree of depth at most d. These left and right subtrees may be chosen independently from the $1 + N_d$ possibilities (the empty subtree is not counted in N_d), so the following recursion relation holds:

$$N_{d+1} = (1 + N_d) \times (1 + N_d).$$

   Thus $N_2 = 25$ and $N_3 = 676$. □

6. **Solution:** Suppose that there are only $T < \infty$ binary trees with n leaves. Pick one with maximal depth d, and find one of its vertices with generation number d. That vertex must be a leaf; adding one daughter to that vertex creates a new binary tree which has n leaves and depth $d+1$, so it cannot be among the original T. Hence there must be at least $T+1$ trees. Hence there cannot be just a finite number. □

7. **Solution:** If there are 10 or fewer letters in the alphabet, each gets a unique one-digit codeword. Otherwise, suppose that the alphabet consists of the consecutive integers $A = \{1, 2, \ldots, N\}$, where $N > 10$. Let the occurrence probabilities be $p_1, p_2, \ldots, p_N$. Then,

   1. Create N orphan vertices with weights $p_1, \ldots, p_N$.
   2. Find the 10 orphans with smallest weights. Label them $x_0, x_1, \ldots, x_9$ and denote their respective weights by $p_{x_0}, \ldots, p_{x_9}$. Use some convention to break ties.
   3. Create a new orphan vertex x, give it weight $p_{x_0} + \cdots + p_{x_9}$, and make it the common parent of vertices x_0 through x_9, which are no longer orphans. Label the edge between x and x_d with the digit d, for $d = 0, 1, \ldots, 9$.

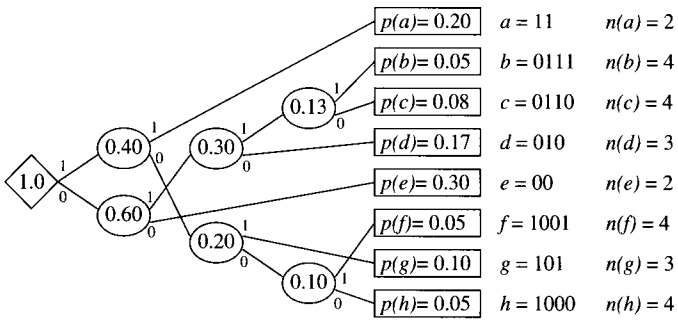


Figure A.5: Huffman code solving Exercise 8.

4. If there are fewer than 10 orphan vertices left, connect them to a common parent, call it the root, and terminate. Otherwise, return to step 2.

□

8. **Solution:** A Huffman code for this combination of alphabet and occurrence probabilities is shown in Figure A.5. It has codeword lengths $M = (2, 4, 4, 3, 2, 4, 3, 4)$, and its bit rate is $\sum_{x \in A} p(x)n(x) = 2.73$, whereas $H(p) = \sum_{x \in A} p(x) \log_2 1/p(x) = 2.69$. □

9. **Solution:** First count the occurrences of each of the 20 letters in this 85-letter message: 1kpw, 2bsv, 3acd, 4fhiy, 5nou, 7l, 8t, 9r, 12e. One Huffman tree for this set of occurrence probabilities is obtained in these steps:

```
1k 1p 1w 2b 2s 2v 3a 3c 3d 4f 4h 4i 4y 5n 5o 5u 7l 8t 9r 12e
1w 2kp 2b 2s 2v 3a 3c 3d 4f 4h 4i 4y 5n 5o 5u 7l 8t 9r 12e
2b 2s 2v 3kpw 3a 3c 3d 4f 4h 4i 4y 5n 5o 5u 7l 8t 9r 12e
2v 3kpw 3a 3c 3d 4bs 4f 4h 4i 4y 5n 5o 5u 7l 8t 9r 12e
3a 3c 3d 4bs 4f 4h 4i 4y 5vkpw 5n 5o 5u 7l 8t 9r 12e
3d 4bs 4f 4h 4i 4y 5vkpw 5n 5o 5u 6ac 7l 8t 9r 12e
4f 4h 4i 4y 5vkpw 5n 5o 5u 6ac 7dbs 7l 8t 9r 12e
4i 4y 5vkpw 5n 5o 5u 6ac 7dbs 7l 8fh 8t 9r 12e
5vkpw 5n 5o 5u 6ac 7dbs 7l 8iy 8fh 8t 9r 12e
5o 5u 6ac 7dbs 7l 8iy 8fh 8t 9r 10vkpwn 12e
6ac 7dbs 7l 8iy 8fh 8t 9r 10ou 10vkpwn 12e
7l 8iy 8fh 8t 9r 10ou 10vkpwn 12e 13acdbs
8fh 8t 9r 10ou 10vkpwn 12e 13acdbs 15liy
9r 10ou 10vkpwn 12e 13acdbs 15liy 16fht
10vkpwn 12e 13acdbs 15liy 16fht 19rou
13acdbs 15liy 16fht 19rou 22vkpwne
16fht 19rou 22vkpwne 28acdbsliy
22vkpwne 28acdbsliy 35fhtrou
35fhtrou 50vkpwneacdbsliy
85fhtrouvkpwneacdbsliy
```

These nodes may be rearranged into a tree with its root at the top:

```
0: 85fhtrouvkpwneacdbsliy
1: 50vkpwneacdbsliy                                  35fhtrou
2: 22vkpwne            28acdbsliy                    16fht       19rou
3: 10vkpwn    12e*     13acdbs       15liy           8fh    8t*  9r*  10ou
4: 5vkpw  5n*          6ac   7dbs    7l*  8iy        4f* 4h*         5o* 5u*
5: 2v* 3kpw         3a* 3c*  3d* 4bs     4i* 4y*
6:    1w* 2kp                  2b* 2s*
7:       1k* 1p*
```

Leaves are marked with asterisks, and the codeword length is listed at left. The canonical Huffman tree produced by this algorithm is described by the maximum codeword length $L = 7$; the number of codewords at each length, $M = (0, 0, 3, 6, 6, 3, 2)$; and the order in which letters appear in the canonical tree at each level, $A' = etr\,nlfhou\,vacdiy\,wbs\,kp$.

It takes 5 bits per letter to code the letters in a 20-letter alphabet, so the original message occupies at least $85 * 5 = 425$ bits.

The bit rate of the Huffman tree is its weighted depth, which is

$$\begin{aligned} &7*(1{+}1) + 6*(1{+}2{+}2) + 5*(2{+}3{+}3{+}3{+}4{+}4) + \\ &\quad + 4*(5{+}7{+}4{+}4{+}5{+}5) + 3*(12{+}8{+}9) \quad = \text{ 346 bits,} \end{aligned}$$

or 4.07 bits per letter.

The entropy $H(p)$ of the message is computed from the occurrence probabilities $p(e) = \frac{12}{85}$, $p(k) = p(p) = p(w) = \frac{1}{85}$, and so on:

$$\begin{aligned} H(p) \quad = \quad & 3 * \frac{1}{85} * \log_2 \frac{85}{1} + 3 * \frac{2}{85} * \log_2 \frac{85}{2} + 3 * \frac{3}{85} * \log_2 \frac{85}{3} \\ & + 4 * \frac{4}{85} * \log_2 \frac{85}{4} + 3 * \frac{5}{85} * \log_2 \frac{85}{5} + 1 * \frac{7}{85} * \log_2 \frac{85}{7} \\ & + 1 * \frac{8}{85} * \log_2 \frac{85}{8} + 1 * \frac{9}{85} * \log_2 \frac{85}{9} + 1 * \frac{12}{85} * \log_2 \frac{85}{12} \\ = \quad & \text{4.02946 bits per letter,} \end{aligned}$$

or about 342.5 total bits. □

10. **Solution:** In this case, adding an extra letter i with occurrence probability 0 deepens the tree by one level. One canonical Huffman code for this appended alphabet and occurrence probabilities is shown in Figure A.6. It has the description $L = 5$, $M = (0, 2, 2, 3, 1)$, and $A' =$eadgcbfh. Its bit rate is $\sum_{x \in A} p(x)n(x) = 2.78$, which is 0.05 bits per character worse than the efficiency of the tree without the extra letter. □

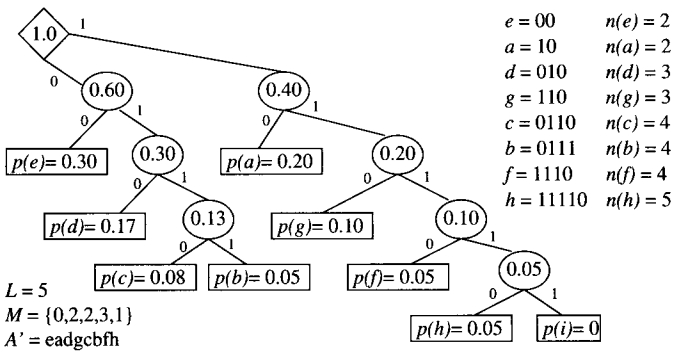


Figure A.6: Canonical Huffman code solving Exercise 10.

11. **Solution:** In this case, $P_{aw} = (1-q)(1-p)^N$, $P_{fa} = q(1-p)^N$, and using the notation of Equation 6.7,

$$\begin{aligned}
P_{de} &= (1-q)\sum_{k\in O}\binom{N}{k}p^k(1-p)^{N-k} + q\sum_{k\in E+}\binom{N}{k}p^k(1-p)^{N-k} \\
&= (1-q)(1-x) + q[x-(1-p)^N] \\
&= \frac{1}{2}[1-(1-2q)(1-2p)^N - 2q(1-p)^N]; \\
P_{ue} &= q\sum_{k\in O}\binom{N}{k}p^k(1-p)^{N-k} + (1-q)\sum_{k\in E+}\binom{N}{k}p^k(1-p)^{N-k} \\
&= q(1-x) + (1-q)[x-(1-p)^N] \\
&= \frac{1}{2}[1+(1-2q)(1-2p)^N - 2(1-q)(1-p)^N].
\end{aligned}$$

□

12. **Solution:** We use Gilbert's ideas from Theorem 6.12:

Gilbert's Algorithm for an Error Correcting Code

```
gilbertcode( bits, words, corrects ):
[0] Let N = 1<<bits, let radius = 2*corrects+1
[1] For n=0 to N-1, allocate tag[n] with tag[n]=LIVE
[2] Allocate c[0],...,c[words-1] and let c[0] = 0
[3] For j=0 to words-1, do [4] to [7]
[4]   For n=c[j]+1 to N-1 do [5]
[5]     If dist(n,c[j])<=radius, then let tag[n] = DEAD
[6]   For n=c[j]+1 to N-1 do [7]
[7]     If tag[n]==LIVE, then let c[j+1]=n and goto [3]
[8] For j=0 to words-1, print c[j]
```

To generate the code, we call `gilbertcode(10,4,2)`. This starts with codeword $\mathbf{c}(0) = 00000\,00000$ and eliminates Hamming spheres of radius $5 = 2 * 2 + 1$ from the ten-dimensional unit hypercube. It continues until it has found three additional survivors $\mathbf{c}(1) = 00001\,11111$, $\mathbf{c}(2) = 01110\,00111$, and $\mathbf{c}(3) = 01111\,11000$. Note that only nine codeword bits are actually needed, as the tenth bit is constant over the code. □

13. **Solution:** If x and y differ at exactly one decimal digit, then $x - y = \pm d \times 10^k$ for some integers $k \geq 0$ and $d \in \{1, 2, \ldots, 9\}$. But $c_{11}(x) = c_{11}(y)$ if and only if $x - y = 0 \pmod{11}$, which requires $d = 0 \pmod{11}$ since $10^k \neq 0 \pmod{11}$ for any $k \geq 0$. But no d in the stated range satisfies that congruence.

    Note that $c_9(110) = c_9(119)$ is a one-digit difference undetected by casting out nines. In general, changing any 0 digit into 9 will not change c_9, since the difference $d = 9$ satisfies $d = 0 \pmod{9}$. Neither will transposing two digits, though transposing unequal adjacent digits will change the value of c_{11}. □

14. **Solution:** No. Consider the counterexample

$$1\,872 \times 22\,883 = 42\,836\,976 \neq 42\,8\hat{4}6\,\hat{8}76,$$

    a two-digit error in decimal arithmetic. We have

$$c_9(1\,872) = 9; \qquad c_9(22\,883) = 5; \qquad c_9(42\,846\,876) = 9 = c_9(9 \times 5),$$

    so c_9 fails to detect the error. Likewise,

$$c_{11}(1\,872) = 2; \qquad c_{11}(22\,883) = 3; \qquad c_{11}(42\,846\,876) = 6 = c_{11}(2 \times 3),$$

    so c_{11} also fails to detect the error. □

15. **Solution:** This is a direct translation of the integer Euclidean algorithm:

    Euclid's Algorithm for Integer-Size Mod-2 Polynomials

```
intmod2polygcd( x, y ):
[0] Let z = x
[1] Call intmod2polydivision(y,x), replace x = remainder
[2] Let y = z
[3] If intmod2polydegree(x)>0, then go to [0]
[4] Return y
```

    □

16. **Solution:** Write $1 = \gcd(x, z) = m_0(t)x(t) + n_0(t)z(t)$ as in the proof of Theorem 1.2. Then $y(t) = m_0(t)x(t)y(t) + n_0(t)z(t)y(t)$. Since $z(t)$ evidently divides $n_0(t)z(t)y(t)$, and $z(t)$ divides $m_0(t)x(t)y(t)$ by assumption, it follows that $z(t)$ divides $m_0(t)x(t)y(t) + n_0(t)z(t)y(t) = y(t)$. □

17. **Solution:** (a) Trial division with all six mod-2 polynomials of degree 1 or 2 yields:

$$\begin{aligned} t^3+t+1 &= (t)(t^2+1)+1 \\ &= (t+1)(t^2+1)+t^2 \\ &= (t^2)(t)+t+1 \\ &= (t^2+1)(t)+1 \\ &= (t^2+t)(t)+t^2+t+1 \\ &= (t^2+t+1)(t)+t^2+1. \end{aligned}$$

All of these expressions have nonzero remainders, so t^3+t+1 is irreducible.

(b) The factorization $t^4+t^2+1=(t^2+t+1)(t^2+t+1)+0=(t^2+t+1)^2$ is discovered by trial division with the six mod-2 polynomials of degree 1 or 2. □

18. **Solution:** For all of these polynomials, we use the following:

Standard C Function: Test Mod-2 Polynomial Divisibility

```
int least_power ( unsigned int mod2poly, int degree ) {
  unsigned int rem, highbit;  int N;
  highbit=0x1<<(degree-1);     /* Start at t^degree +1 */
  for(rem=mod2poly, N=degree;  rem!=0x1;  ++N )
    if( rem & highbit ) {
      rem = (rem<<1)^mod2poly; /* shift, add mod2poly */
    } else
      rem <<= 1;  /* else just leftshift all the bits */
  return N;
}
```

(a) Call `least_power()` with `mod2poly` set to 1011 (base 2) and `degree` set to 3. The return value shows that t^3+t+1 divides t^N+1, with $N=7=2^3-1$, but no smaller $N>0$.

(b) Mod-2 polynomial 1100000001111, of degree 12, divides t^N+1 for $N=2047=2^{11}-1$, but no smaller $N>0$.

(c) Mod-2 polynomial 1100000000000101, of degree 16, divides t^N+1 for $N=32767=2^{15}-1$, but no smaller $N>0$.

(d) Mod-2 polynomial 110000000010100010000001, of degree 24, divides t^N+1 for $N=7161$, but no smaller $N>0$. □

19. **Solution:** On a computer that has integer types with 33 or more bits, we can use `least_power()` exactly as in Solution 18. Otherwise, on a computer with 32-bit integers, we simply remove the leftmost, most significant, 33rd bit, setting `mod2poly` to 00000100110000010001110110110111 (base 2), and

call `least_power()` with that and the same value 32 for `degree`. Integer overflow eliminates the highest order bit in the remainder computation, and after a rather long while we get the result. □

20. **Solution:** The encoding, formatted as 8-bit codewords with spaces for readability, is the following:

```
11000101 01101100 01100101 11110000 11101000 11100001 11101110 01110100
11110011 10100000 11100001 01110010 01100101 10100000 11100001 11110000
11110000 01110010 01101111 11100001 01100011 11101000 01101001 11101110
11100111 10100000 01100110 01110010 01101111 11101101 10100000 01110100
11101000 01100101 10100000 01010011 01101111 11110101 01110100 11101000
00100001 (base 2)
```

The conversion to bits with the addition of a parity bit was done on a computer with ASCII internal coding by the following function, called with `msg[]` set to "Elephants are approaching from the South!"

Standard C Function: Write ASCII+Parity Bits

```
int ascii_bits(int allbits[], const char msg[] ) {
  int i, n=0, b, letter, parity, bit[8];
  for(i=0; letter=msg[i]; i++) { /* Stop at NULL */
    for(parity=0, b=6;  b>=0;  b--) {
      bit[b] = (letter>>b)&1;    /* Assume ASCII */
      parity += bit[b];
    }
    bit[7] = parity&1; /* preserve even parity */
    for(b=7; b>=0; b--) allbits[n++] = bit[b];
  }
  return n;             /* Number of bits written */
}
```

□

21. **Solution:** We first call the function `ascii_bits()` from the previous exercise with `msg` set to "Elephants are approaching from the North!" to get the following output:

```
11000101 01101100 01100101 11110000 11101000 11100001 11101110 01110100
11110011 10100000 11100001 01110010 01100101 10100000 11100001 11110000
11110000 01110010 01101111 11100001 01100011 11101000 01101001 11101110
11100111 10100000 01100110 01110010 01101111 11101101 10100000 01110100
11101000 01100101 10100000 01001110 01101111 01110010 01110100 11101000
00100001  (base 2)
```

Next, we implement `mod2polychecksum()` on page 217 as follows:

Standard C Function: Mod-2 Polynomial Modular Checksum

```
int mod2polychecksum( int msgbit[], int numbits,
                      int mod2poly[], int deg ) {
  int checksum, d, n;
  for(n=0; n<numbits-deg; n++ )
    if( msgbit[n]==1 )
      for(d=0; d<=deg; d++)
        msgbit[n+d] ^= mod2poly[deg-d];
  for(checksum=0; n<numbits; n++)
    checksum = 2*checksum + msgbit[n];
  return checksum;
}
```

We will use `mod2poly[]={1,0,1,1}`, representing t^3+t+1 with the coefficient of t^3 at index 0. Applying this with `msgbit[]` containing the encoded bits from the message "Elephants are approaching from the South!" as computed in Solution 20, and putting the left-most bit of "E" at index 0, yields a checksum of 5.

Calling `mod2polychecksum()` with the "...North!" base-2 bits in the array `msgbit[]` and the same mod-2 modulus polynomial yields a checksum of 5, since the many differences happen to cancel. However, "Elephants are approaching from the NORTH!" yields the following bitstring:

```
11000101 01101100 01100101 11110000 11101000 11100001 11101110 01110100
11110011 10100000 11100001 01110010 01100101 10100000 11100001 11110000
11110000 01110010 01101111 11100001 01100011 11101000 01101001 11101110
11100111 10100000 01100110 01110010 01101111 11101101 10100000 01110100
11101000 01100101 10100000 01001110 01101111 01110010 01110100 11101000
00100001  (base 2)
```

`mod2polychecksum()` returns its checksum as 0. Thus, this message is distinguished from the first by its 3-bit checksum, but the previous one is not. □

Appendix B

Basics, Technicalities, and Digressions

B.1 ASCII and other character sets

The *American Standard Code for Information Interchange*, variously called ASCII, US-ASCII, ANSI X3.4-1968, or ISO 646, assigns a number in the range 0 to 127 to each of the common English-language typewriter symbols plus some nonprinting *control characters*. It is a decades-old standard, still commonly used to represent text within computers, even in non-English-speaking countries. Table B.1 gives the ASCII character numberings in decimal, octal, and hexadecimal notation.

The caret (^) preceding a character means "hold the Control key while entering." The low code numbers 0–31 plus the highest code number 127 are *control characters* that have various functions. For example `^G`, read "Control-G," rings the margin bell on a teletype, or causes a computer monitor to beep. Codes 20–126 are letters, numbers, and punctuation, with code 20 being the space character.

The Standard C programming language does not depend on ASCII numbering, but it uses all the ASCII letters, numbers, and punctuation symbols except codes 24 ($), 40(@), and 60(`). Standard C also requires certain of the control characters, referring to them by *escape sequences* beginning with a backslash (\). These are listed in Table B.2.

Because of the way C handles strings, every character numbering scheme must have the same internal representation, 0, for the Null character. Also, Standard C requires that the internal codes for 0,1,...,9 have sequential values.

ASCII codes fit into 7 bits, but most computers use 8 bits per character. Thus, there are various extended ASCII character sets, with codes 128–255 used for accented letters, graphical symbols, Greek letters, or other purposes. Symbols represented by these codes are sometimes called *meta-characters*. There is a growing family of international standard alphabets using 8-bit character codes, the ISO 8859-n family of national character sets, of which the first ten members are listed

Char	Hex	Oct	Dec	Char	Hex	Oct	Dec	Char	Hex	Oct	Dec	Char	Hex	Oct	Dec
^@	00	000	0		20	040	32	@	40	100	64	‘	60	140	96
^A	01	001	1	!	21	041	33	A	41	101	65	a	61	141	97
^B	02	002	2	"	22	042	34	B	42	102	66	b	62	142	98
^C	03	003	3	#	23	043	35	C	43	103	67	c	63	143	99
^D	04	004	4	$	24	044	36	D	44	104	68	d	64	144	100
^E	05	005	5	%	25	045	37	E	45	105	69	e	65	145	101
^F	06	006	6	&	26	046	38	F	46	106	70	f	66	146	102
^G	07	007	7	’	27	047	39	G	47	107	71	g	67	147	103
^H	08	010	8	(	28	050	40	H	48	110	72	h	68	150	104
^I	09	011	9	)	29	051	41	I	49	111	73	i	69	151	105
^J	0A	012	10	*	2A	052	42	J	4A	112	74	j	6A	152	106
^K	0B	013	11	+	2B	053	43	K	4B	113	75	k	6B	153	107
^L	0C	014	12	,	2C	054	44	L	4C	114	76	l	6C	154	108
^M	0D	015	13	-	2D	055	45	M	4D	115	77	m	6D	155	109
^N	0E	016	14	.	2E	056	46	N	4E	116	78	n	6E	156	110
^O	0F	017	15	/	2F	057	47	O	4F	117	79	o	6F	157	111
^P	10	020	16	0	30	060	48	P	50	120	80	p	70	160	112
^Q	11	021	17	1	31	061	49	Q	51	121	81	q	71	161	113
^R	12	022	18	2	32	062	50	R	52	122	82	r	72	162	114
^S	13	023	19	3	33	063	51	S	53	123	83	s	73	163	115
^T	14	024	20	4	34	064	52	T	54	124	84	t	74	164	116
^U	15	025	21	5	35	065	53	U	55	125	85	u	75	165	117
^V	16	026	22	6	36	066	54	V	56	126	86	v	76	166	118
^W	17	027	23	7	37	067	55	W	57	127	87	w	77	167	119
^X	18	030	24	8	38	070	56	X	58	130	88	x	78	170	120
^Y	19	031	25	9	39	071	57	Y	59	131	89	y	79	171	121
^Z	1A	032	26	:	3A	072	58	Z	5A	132	90	z	7A	172	122
^[	1B	033	27	;	3B	073	59	[	5B	133	91	{	7B	173	123
^\	1C	034	28	<	3C	074	60	\	5C	134	92	|	7C	174	124
^]	1D	035	29	=	3D	075	61	]	5D	135	93	}	7D	175	125
^^	1E	036	30	>	3E	076	62	^	5E	136	94	~	7E	176	126
^_	1F	037	31	?	3F	077	63	_	5F	137	95	^?	7F	177	127
Char	Hex	Oct	Dec	Char	Hex	Oct	Dec	Char	Hex	Oct	Dec	Char	Hex	Oct	Dec

Table B.1: ASCII characters and their numerical codes

Control function	ANSI Standard C escape sequence	ASCII name	ASCII hex	ASCII octal	ASCII decimal
Null	\0	Control-@	00	000	0
Alert	\a	Control-G	07	007	7
Backspace	\b	Control-H	08	010	8
Horizontal tab	\h	Control-I	09	011	9
New line	\n	Control-J	0A	012	10
Vertical tab	\v	Control-K	0B	013	11
Form feed	\f	Control-L	0C	014	12
Carriage return	\r	Control-M	0D	015	13

Table B.2: ASCII implementation of required control codes in Standard C

ISO Number	Name	Examples
ISO-8859-1	Latin1	Basque, English, French, Swedish
ISO-8859-2	Latin2	Croatian, Hungarian, Polish, Sorbian
ISO-8859-3	Latin3	Esperanto, Maltese
ISO-8859-4	Latin4	Estonian, Lappish, Latvian, Lithuanian
ISO-8859-5	Cyrillic	Bulgarian, Macedonian, Russian
ISO-8859-6	Arabic	Basic Arabic
ISO-8859-7	Greek	Modern Greek
ISO-8859-8	Hebrew	Hebrew, Yiddish
ISO-8859-9	Latin5	Turkish
ISO-8859-10	Latin6	Icelandic, Inuit

Table B.3: The first ten ISO-8859 national character sets

in Table B.1. In each of these, the first 128 symbols are the same as ASCII.

There are also several using 16-bit *wide characters*, such as *Unicode*, or ISO 10646, which encompasses all of Planet Earth's written human languages.

B.2 Algorithms

An *algorithm* is a step-by-step procedure for computing some quantity. It has *inputs*, *outputs*, and an ordered list of *instructions*. An algorithm is *finite* if, for any fixed finite inputs, each of its steps is finite and the total number of steps performed is also finite.

Notation for the instructions must be unambiguous. For this purpose there exist numerous programming languages with limited vocabulary and rigid grammar and syntax. A nice survey of these, including old and exotic ones like APL, may be found in Tucker's *Programming Languages*. Algorithms in this text are written in *pseudocode* that resembles some of these formal programming languages, especially Basic, FORTRAN and Standard C. They are aimed at humans and are not suitable for direct machine consumption. However, we must still have some rules to eliminate ambiguity.

If intermediate quantities are computed and stored, they must be assigned to *memory*. The equals sign is used in this context, and another symbol "==" is used in tests for equality. We may also use the APL assignment arrow $x \leftarrow x + 1$, read "x gets x plus 1," to signify that the memorized quantity x should be replaced by itself added to 1.

Data type and variable declarations are omitted in pseudocode, and there is no FORTRAN-like naming convention to distinguish integers from floats, but new quantities and their types should be inferred from context. Function names will always be in lower-case, followed by parentheses enclosing a parameter list. When a function is defined for the first time, there will be a colon after the parameter list to signify the beginning of the definition. Punctuation marks will also be used

to signify parts of data structures. Keywords and action descriptions will either be capitalized or all lower-case, but can be distinguished from function names by the absence of parentheses.

Array indexing is signified by placing the index inside square brackets. Members of data structures are set apart with periods. Operations are always evaluated from left to right, so that `CW[J].VAL = X-Y-Z` means "assign the value of $(x - y) - z$ to member `VAL` of the data structure at index j in the array of data structures `CW[]`." Notice how arrays are marked by appending an empty set of brackets. An array of arrays can be denoted by two sets of empty brackets: `MATRIX[][]`. Likewise, writing `sqrt()` emphasizes that a function may have a nonempty list of parameters, though at the moment those parameters are not the focus of attention.

The operators `%`, `<<`, `>>`, `&`, `|`, and `^`, taken from Standard C, mean remainder, left-shift, right-shift, bitwise conjunction, bitwise disjunction, and bitwise exclusive-or, respectively. Following any of these with `=` means "assign to the left-hand variable the output of the operation." For example, `x%=y`, means "assign to `x` the result of `x%y`," and so on.

Likewise, the Standard C logical operators `!`, `&&`, `||`, `==` and `!=` will be used to signify NOT, AND, OR, EQUAL, and NOT EQUAL, respectively. We will assume that logical expressions have the values TRUE or FALSE that can be tested, though unlike Standard C we will not give them arithmetic values or compute with them.

Algorithms are set apart from the text in numbered lists. Statements and keywords will be similar to those in the following list of examples:

Assignment: `Let x = 0`, or $x \leftarrow 0$, assigns the value "0" to the memory location labeled x.

Function calls: `Compute x = sum(arr[],len)`, or $x \leftarrow \text{sum}(arr, len)$, evaluates the function `sum()` with input values `arr[]` and `len`, then assigns its return value to `x`.

Replacement: Operations sometimes perform arithmetic with a variable and then replace that variable with the result:

Increments: `x += dx` means "$x \leftarrow x + dx$".

Decrements: `y -= dy` means "$y \leftarrow y - dy$".

Multipliers: `z *= c` means "$z \leftarrow c \times z$".

Normalizers: `w /= sum` means "$w \leftarrow w/sum$".

The words "Replace," "Increment," "Accumulate," "Decrement," "Multiply," "Normalize," or "Divide" may be used to emphasize the nature of the arithmetic.

Conditionals: The logical expression between "If" and "then" is tested. If its value is TRUE, the "then" action is performed. Otherwise, the "Else" subsequent action is performed, if there is one. After either case, the algorithm moves past both the "If" and the "Else" lines. For example, the following

fragment assigns $y = 2$ at the "If" line if $x == 4$ and assigns $y = 3$ at the "Else" line if $x == 5$:

```
[0] If x is even, then let y = x/2
[1] Else let y = (x+1)/2
```

Loops: These contain a control statement at the top, and a *body* consisting of other instructions grouped below it and indicated by indentation.

In a "for" loop, the control statement names and defines the range of the loop index, which is incremented by +1 at each pass through the body:

```
[0] For j=a to b, do [1] to [2]
[1]   Let array[2*j] = j
[2]   Let array[2*j+1] = 0
```

If $b < a$, then no statement in the body of the loop will be executed.

In a "while" loop, the loop index may be altered in any desired fashion within the body:

```
[0] Let k = n
[1] While k>0, do [2] and [3]
[2]    Multiply nfacfac *= k
[3]    Decrement k -= 2
```

Return statements: These indicate the value to be returned by a function:

```
...
[4] Return nfacfac
```

If no value is named it means that the function acts solely by *side-effect*, and returns no explicit output. The return statement can be omitted in this case.

Predefined functions: It will be assumed that common functions are predefined, such as `sqrt(x)`, `sin(x)`, `cos(x)`, `exp(x)`, and `log(x)`, which respectively return the square root, sine, cosine, exponential and logarithm of x. Exceptions such as $x \leq 0$ in `log(x)` cause immediate termination.

B.3 Big-Oh notation

So-called "big-Oh" notation is used to estimate the rate of increase of a quantity that depends on a parameter. For example, suppose that t_n is the number of arithmetic operations needed to compute the square of an n-digit integer on a particular machine. To say that $t_n = O(n^2)$ means there is some finite positive constant C such that, for all $n = 1, 2, \ldots$ and any n-digit number, $t_n \leq Cn^2$. The smallest value of C which works is called the *sharp* constant and is sometimes of independent interest. In this example, the constant depends on the base of

the number system used and certain other details about the machine arithmetic, unimportant particulars best hidden by the notation.

For example, if a and b are n-digit integers, then the naive paper-and-pencil algorithms for computing $a + b$, ab, and a/b require $O(n)$, $O(n^2)$, and $O(n^2)$ one-digit operations, respectively. These are finite numbers of operations for any fixed a, b on a particular machine.

The complexity of an algorithm containing nested loops can be estimated from the depth of the nesting. For example, $n \times n$ matrix multiplication has three nested loops each requiring $O(n)$ iterations, and takes $O(n^3)$ total operations.

Other increasing functions of n may be used to estimate complexity. For example, the complexity of printing the number n grows like $O(\log n)$, since n has $O(\log n)$ digits and printing each one should take at most some fixed amount of time. Not specifying the constant allows the estimate to apply to printing in all bases, on any kind of printer.

The complexity of a combination of algorithms is determined by the most complex component. For example, solving the $n \times n$ linear system $\mathbf{Ax} = \mathbf{b}$ is a combination of inverting matrix $\mathbf{A}$, which costs $O(n^3)$ operations, followed by applying the inverse matrix to the vector $\mathbf{b}$, which costs $O(n^2)$. The total complexity is $O(n^3)$, since for all large enough n the inversion cost will dominate. In general, a $O(n^p)$ algorithm combined with a $O(n^q)$ algorithm has complexity $O(n^{\max(p,q)})$.

Memory space requirements for an algorithm can be specified in big-Oh notation as well. For example, to store an $n \times n$ image as pixel values will cost $O(n^2)$ memory spaces. The notation hides the details of how many bits are stored per pixel, in the unspecified constant C. Space and time are both important in complexity estimates, and an algorithm that performs $O(n)$ operations on each pixel will cost $O(n) \times O(n^2) = O(n^3)$ operations to execute. In general, $O(n^p)$ operations on each of $O(n^q)$ values will cost $O(n^{p+q})$ in total.

The same notation is used to give estimates of small quantities in terms of a small parameter. For example, suppose that $f = f(x)$ is a real-valued function, defined for values x near 1. To say that $f(1+h) - f(1) = O(h)$ means that there is some constant C such that $|f(1+h) - f(1)| \leq C|h|$ for all h. If we only care about very small h, we might say $f(1+h) - f(1) = O(h)$ as $h \to 0$, which is the weaker assertion that for some $\epsilon > 0$, there is a constant C such that $|f(1+h) - f(1)| \leq C|h|$ for all h with $|h| < \epsilon$.

Size estimates can be combined just like complexity estimates, with the higher order being dominant. For example, if $f(h) = O(h^p)$ and $g(h) = O(h^q)$ as $h \to 0$, then $f(h) + g(h) = O(h^{\max(p,q)})$ as $h \to 0$. Also, $f(h)g(h) = O(h^{p+q})$ as $h \to 0$.

B.4 Calculus methods

Complete proofs of the lemmas and theorems in this section may be found in Apostol's two-volume *Calculus*, listed in the further readings, or else in his more advanced book *Mathematical Analysis*, listed in the further readings in Chapter 3.

B.4.1 Techniques of proof

Suppose that $P(n)$ is a property of the counting number $n \in \{0, 1, 2, \ldots\}$, such as "$n$ is odd" or "the sum of the positive integers up to n is $n(n+1)/2$." $P(n)$ is considered *verifiable* if it can be shown true or false with only finitely many calculations. Both of the example properties are verifiable: the first by dividing n by 2, the second by performing n additions.

The verifiability of $P(n)$ for any fixed n sometimes permits verifying that $P(n)$ is true for all the infinitely many n, using the *principle of mathematical induction.* If $P(0)$ is true, and if it can be verified algebraically for the variable m that $P(m+1)$ is true whenever $P(m)$ is true, then $P(n)$ is true for every $n = 0, 1, 2, \ldots$. Any doubter worried about a particular n need only consider that $P(0)$ is true, so $P(1)$ is true, so $\ldots P(n-1)$ is true, and finally $P(n)$ is true.

For the first example, $P(0)$ is false (since 0 is even, not odd) so induction fails at the outset. The second property is true for all n, since $P(0)$ is true (the sum of no integers is $0 = (0)(0+1)/2$) and $P(m+1)$ follows algebraically from $P(m)$:

$$\sum_{k=1}^{m+1} k = (m+1) + \sum_{k=1}^{m} k.$$

If $P(m)$ is true, then the last sum is $m(m+1)/2$. Combining this algebraically with $(m+1)$ gives $\sum_{k=1}^{m+1} k = (m+1)(m+2)/2$, or $P(m+1)$.

Sometimes it is more convenient to start with $P(1)$ or higher, if the resulting algebraic formulas are nicer.

Enumeration

To show that a set is infinite, it suffices to show that it contains at least n members for every integer n. This can be done with mathematical induction, with $P(n)$ being the proposition that the set contains at least n members. Euclid first used this idea to show that there are infinitely many primes. $P(1)$ is true, since 2 is prime. If $P(n)$ is true, let $p_1, \ldots, p_n$ be n distinct primes and put $x = p_1 \cdots p_n + 1$. Then x is not divisible by any of the n listed primes, so it is either a new prime number or contains a prime divisor not in the list, so that $P(n+1)$ is true.

Polynomial algebra

Let $p = p(x)$ be a polynomial with coefficients $a_0, a_1, \ldots, a_n$:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n. \tag{B.1}$$

The degree of p is the nonnegative integer n except when p is identically zero, that is, when $a_0 = a_1 = \cdots = a_n = 0$. In that case, the degree is conventionally treated as $-\infty$ with the arithmetic rules $n + (-\infty) = (-\infty) + n = -\infty$ for any finite n.

If $p = p(x)$ and $q = q(x)$ are polynomials of degrees n, m respectively, then

- Both $pq = p(x)q(x)$ and $p + q = p(x) + q(x)$ are polynomials;

- The degree of pq is $n + m$;
- The degree of $p + q$ is at most $\max\{n, m\}$;
- If q is not identically zero and thus has finite degree m, there are unique polynomials $r = r(x)$ and $s = s(x)$, with the degree of s being less than m, such that for all x,

$$p(x) = s(x)q(x) + r(x).$$

The last fact implies that if z is a root of p, then $p(x) = s(x)(x - z)$ for all x. We prove this by dividing p by $q(x) = x - z$, which has degree $m = 1$. This gives a remainder polynomial $r = r(x)$ of degree 0 or less. But then r is constant, and since $0 = p(z) = s(z)(z - z) + r(z) = r(z)$, we conclude that r is identically zero.

B.4.2 Limits, continuity, and derivatives

Let $f = f(t)$ be a function of one real variable, defined for all values of t in an open interval (a, b). Let t_0 signify some particular fixed value of the variable t, and let ϵ and δ signify small positive real numbers.

Function f is said to *have a limit* L at $t = t_0$ if, for every $\epsilon > 0$, there is a $\delta > 0$ such that if $0 < |t - t_0| . < \delta$, then $|f(t) - L| < \epsilon$. That fact about f is often abbreviated by the notation

$$\lim_{t \to t_0} f(t) = L.$$

It does not require that f be defined at t_0, but only that it be defined at points t near t_0, and that its value at those points stays close to L.

Sometimes a function has a limit from only one side. For example, f is said to *have a left sided limit* L at $t = t_0$ if, for every $\epsilon > 0$, there is a $\delta > 0$ such that if $t_0 - \delta < t < t_0$, then $|f(t) - L| < \epsilon$. Such a limit L, if it exists, is usually denoted by $f(t_0-)$ as follows:

$$\lim_{t \to t_0-} f(t) = f(t_0-).$$

This does not require that f be defined at any $t \geq t_0$.

Likewise, f is said to *have a right sided limit* L at $t = t_0$ if, for every $\epsilon > 0$, there is a $\delta > 0$ such that if $t_0 < t < t_0 + \delta$, then $|f(t) - L| < \epsilon$. Such a limit L, if it exists, is usually denoted by $f(t_0+)$ as follows:

$$\lim_{t \to t_0+} f(t) = f(t_0+).$$

This does not require that f be defined at any $t \leq t_0$.

If f is defined only on the half-open interval $[a, b)$, then it can at best have a right sided limit at a because its values are undefined at $t < a$. Note that f might have a left sided limit at b in this case, even though it is not defined at b.

Lemma B.1 *f has a limit L at t_0 if and only if its left sided and right sided limits exist and agree: $f(t_0-) = f(t_0+) = L$.* □

Function f is said to be *continuous at a point* t_0 if $\lim_{t \to t_0} f(t) = f(t_0)$. This requires both that the limit exists and that f is defined at $t = t_0$.

Corollary B.2 *f is continuous at t_0 if and only if $f(t_0-) = f(t_0) = f(t_0+)$.* □

Continuity is a concept that applies to functions of more than one variable. A function $f = f(\mathbf{x})$ of the variables $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$ is said to be *continuous at a point* $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbf{R}^n$ if $\lim_{\mathbf{x} \to \mathbf{z}} f(\mathbf{x}) = f(Z)$. Here $\mathbf{x} \to \mathbf{z}$ means that $x_k \to z_k$ for all $k = 1, 2, \ldots, n$.

Thus, for any subset $I \subset \mathbf{R}^n$, we may say that f is *continuous on a set* I if it is continuous at each point of I. We may further say that f is *uniformly continuous on a set* I if the same $\delta(\epsilon)$ relation works at each point of I: for every $\epsilon > 0$, there is a $\delta > 0$ such that $|f(\mathbf{x}) - f(\mathbf{z})| < \epsilon$ for all $\mathbf{x}, \mathbf{z} \in I$ satisfying $|z_k - x_k| < \delta$ for all $k = 1, \ldots, n$.

The behavior of any continuous f has additional restrictions when the set $I \subset \mathbf{R}^n$ is closed, bounded, or connected:

Lemma B.3 *If f is continuous on a closed and bounded set $I \subset \mathbf{R}^n$, then f is uniformly continuous on I.* □

Lemma B.4 *If f is a continuous real-valued function on a closed and bounded set $I \subset \mathbf{R}^n$, then f takes its minimum and maximum values at some points in I.* □

Theorem B.5 (Intermediate Value Theorem) *If f is a continuous real-valued function on a closed and bounded interval $I = [a, b] \subset \mathbf{R}$, and y is any number between $f(a)$ and $f(b)$, then there is some $x \in I$ such that $f(x) = y$.* □

Corollary B.6 *If f is continuous on $[a, b] \subset \mathbf{R}$, then $\{f(x) : a \leq x \leq b\}$ is a closed and bounded interval.* □

Differentiability, or smoothness, is better than continuity. A function f of one real variable is said to be *differentiable at a point* t_0 if the difference quotient function $[f(t) - f(t_0)]/(t - t_0)$ has a limit as $t \to t_0$. The value of this limit is denoted by $f'(t_0)$, and called the *derivative* of f at t_0:

$$f'(t_0) = \lim_{t \to t_0} \frac{f(t) - f(t_0)}{t - t_0}.$$

Lemma B.7 *If f is differentiable at t_0, then f is continuous at t_0.* □

The last lemma holds because if f is differentiable at t_0, then $f(t) - f(t_0)$ must go to zero "at least proportionally as fast" as $t - t_0$ goes to zero as $t \to t_0$. This leads to a notation for comparing rates at which functions approach limits:

A function $g = g(h)$ is said to be $O(h)$, or *big Oh of* h, as $h \to 0$, if there is some proportionality constant C such that $|g(h)| \leq C|h|$ for all h sufficiently close to 0. We may ignore $h = 0$ if $g(0)$ is undefined. This notion can be generalized to $O(h^2)$ and higher powers of h.

Notice that if f is differentiable at t_0, then $g(h) = f(t_0 + h) - f(t_0) = O(h)$. However, even if $f(t) - f(t_0) = O(t - t_0)$, it does not imply that f is differentiable at t_0. It does imply that f is continuous at t_0; in fact, it defines what is meant by *Lipschitz continuity* of f at t_0, which is a stronger notion. All Lipschitz continuous functions are continuous, but some continuous functions (such as $f(t) = t^{1/3}$ at $t_0 = 0$) are not Lipschitz continuous. Another notion of approach rate is needed, so we will say a function $g = g(h)$ is $o(h)$, or *little oh of* h, as $h \to 0$ if

$$\lim_{h \to 0} g(h)/h = 0.$$

In other words, $g(h)$ should go to zero somewhat faster than h. This notion can also be generalized to higher powers of h.

The little-oh notation gives a useful characterization of differentiability:

Theorem B.8 *f is differentiable at t_0 if and only if there is a number, which shall be called $f'(t_0)$, such that $f(t) = f(t_0) + (t - t_0)f'(t_0) + o(t - t_0)$ as $t \to t_0$.* □

A function f is said to be *differentiable on the interval* (a, b) if it is differentiable at each point $t_0 \in (a, b)$. The function f', whose value at each $t_0 \in (a, b)$ is $f'(t_0)$, is called the *derivative* of f.

The Mean Value Theorem is useful for estimating and approximating quantities. It has both a differential and an integral form:

Theorem B.9 (MVT-D) *Suppose that $f = f(x)$ is differentiable on the interval (a, b) and is continuous at the endpoints a, b as well. Then there is some $z \in (a, b)$ such that $f'(z) = [f(b) - f(a)]/(b - a)$.* □

Theorem B.10 (MVT-I) *Suppose that $g = g(x)$ is a continuous function on the interval $[a, b]$. Then there is some $z \in (a, b)$ such that $\int_a^b g(x)\,dx = (b - a)g(z)$.* □

Higher order derivatives of f, if they exist, are obtained by the rule

$$f^{(d+1)}(t) = (f^{(d)}(t))', \qquad \text{for } d = 1, 2, \ldots, \tag{B.2}$$

where $f^{(d)}$ is shorthand for f followed by d apostrophes or primes. For example, $f(t) = \sin 2t$ has $f'(t) = 2\cos 2t$, $f''(t) = -4\sin 2t$, $f'''(t) = -8\cos 2t$, and $f^{(4)}(t) = 16\sin 2t$. By convention, $f^{(0)}$ means f.

Function f is said to be *d-times continuously differentiable*, for $d \geq 1$, if f and its derivative functions $f', f'', \ldots, f^{(d)}$ are all continuous. For such f, we write $f \in C^d$. For example, $f(t) = \sin 2t$ belongs to C^∞, while $f(t) = t^{7/3}$ belongs to C^2 but not to C^3. By convention, we write $f \in C^0$ if f is merely continuous. Also, if $f \in C^d$ for all $d = 1, 2, \ldots$, we write $f \in C^\infty$.

B.4.3 Convergence of sequences, series and products

An *infinite sequence* $\{a(n) : n = 1, 2, \ldots\}$ is a function defined on $\mathbf{Z}^+$. It may take values in $\mathbf{R}$ or $\mathbf{C}$. In both cases, the absolute value $|a(n) - L|$ measures the distance between some term $a(n)$ and some number L.

Infinite sequence $\{a(n) : n \in \mathbf{Z}^+\}$ is said to *converge* if there is some number L such that, for every $\epsilon > 0$, there is some sufficiently large $N = N(\epsilon)$ for which $n \geq N \Rightarrow |a(n) - L| < \epsilon$. In that case, $\{a(n)\}$ is said to have the *limit* L as $n \to \infty$, and we write $\lim_{n\to\infty} a(n) = L$.

The *partial sums sequence* of $\{a(n) : n \in \mathbf{Z}^+\}$ is another infinite sequence $\{s(N) : N \in \mathbf{Z}^+\}$ whose terms are

$$s(N) \stackrel{\text{def}}{=} \sum_{n=1}^{N} a(n), \qquad \text{for } N = 1, 2, 3, \ldots$$

These take values in the same set, $\mathbf{R}$ or $\mathbf{C}$, as $\{a(n)\}$.

Now suppose that $\{a(n)\}$ is an infinite sequence. We say that it is *(conditionally) summable* if its partial sums converge, namely $\lim_{N\to\infty} \sum_{n=1}^{N} a(n)$ exists. We say that it is *absolutely summable* if the partial sums of $\{|a(n)|\}$ converge, namely $\lim_{N\to\infty} \sum_{n=1}^{N} |a(n)|$ exists. We say it is *square-summable* if the partial sums of $\{|a(n)|^2\}$ converge, namely $\lim_{N\to\infty} \sum_{n=1}^{N} |a(n)|^2$ exists.

An absolutely summable sequence can be added up on a finite precision machine in any order, always yielding the same sum except for rounding error. For a summable but not absolutely summable series, order matters: rearrangement might even make the partial sums diverge, even in exact arithmetic. This is Riemann's Theorem on conditionally summable series, page 197 of Apostol's *Mathematical Analysis*. Every absolutely summable sequence is summable and square-summable, but $\{(-1)^n/n : n \in \mathbf{Z}^+\}$ is summable to $\log \frac{1}{2}$ and square-summable to $\pi^2/6$ without being absolutely summable.

A *doubly infinite sequence* is a real or complex valued function defined on $\mathbf{Z}$ rather than $\mathbf{Z}^+$. It can be denoted by $\{a(n) : n = 0, \pm 1, \pm 2, \ldots\}$ and considered as two infinite sequences. It is called summable if both the positive-indexed and negative-indexed sides individually are summable. Likewise, it is called square-summable or absolutely summable if both sides have the corresponding property.

Sequences of functions

A sequence of functions $\{u_n = u_n(t)\}$ is said to *converge at a point* t_0 if the sequence of numbers $\{u_n(t_0) : n = 1, 2, \ldots\}$ converges as $n \to \infty$. If $\{u_n(t)\}$ converges at each t, then the limits define a function $u(t) = \lim_{n\to\infty} u_n(t)$. The convergence is said to be *uniform* if $u_n(t) \to u(t)$ at comparable rates for each t, that is, if for every $\epsilon > 0$ there is a sufficiently large N such that $n \geq N \Rightarrow |u_n(t) - u(t)| < \epsilon$ for every t.

Lemma B.11 *If u_n is a continuous function for each n and $u_n(t) \to u(t)$ converges uniformly in t, then $u = u(t)$ is a continuous function.* □

This result is proved as Theorem 9.2 of Apostol's *Mathematical Analysis*. It implies that if $\{v_n = v_n(t)\}$ is a sequence of bounded continuous functions and $\{c_n\}$ is an absolutely summable sequence, then $\sum c_n v_n$ is a continuous function.

One easy way to prove uniform convergence is to compare with an absolutely summable sequence of numbers:

Theorem B.12 (Weierstrass M-test) *Suppose $\{f_n = f_n(t) : n \in \mathbf{Z}^+\}$ is a sequence of functions defined on some domain $D \subset \mathbf{R}$, and there is a sequence of nonnegative upper bounds $\{M_n : n \in \mathbf{Z}^+\}$ satisfying $|f_n(t)| \leq M_n$ for all $n \in \mathbf{Z}^+$ and all $t \in D$. If $\sum_n M_n$ converges, then*

1. *For each $t \in D$, $\sum_n f_n(t)$ converges absolutely;*
2. *$\sum_n f_n$ converges uniformly on D;*
3. *$|\sum_n f_n(t)| \leq \sum_n M_n$ for every $t \in D$.*

Proof: See Apostol's *Mathematical Analysis*, Theorem 9.6, page 223. □

Rapid decrease

A function $f = f(x)$ of one real variable is said to have *order-d decrease at infinity* if $|f(x)| = O(1/|x|^d)$ as $x \to \pm\infty$, namely, if there are positive constants r_d and C_d, independent of x, such that $|f(x)| < C_d/|x|^d$ whenever $|x| > r_d$. For example, $f(x) = 1/(1+x^2)$ has order-2 decrease at infinity.

A function may have order-d decrease at infinity for more than one d, proved using different constants r_d and C_d. The largest d that works is called the *order of decrease* of f, but f is said to have *rapid decrease at infinity* if it has order-d decrease at infinity for every integer $d > 0$. For example, $f(x) = e^{-(\log|x|)^2}$ has rapid decrease at infinity.

A stronger condition than rapid decrease is *exponential decrease at infinity*. There must be positive constants ϵ, r and C, independent of x, such that $|f(x)| < Ce^{-\epsilon|x|}$ for all $|x| > r$. For example, $f(x) = e^{-x^2}$ has exponential decrease at infinity, whereas $e^{-(\log|x|)^2}$ does not. Exponential decrease at infinity implies rapid decrease.

We also distinguish order of decrease as $x \to \infty$ from decrease as $x \to -\infty$.

Infinite products

The *infinite product* of a sequence $\{b(n) : n = 1, 2, \ldots\}$ is defined by

$$\prod_{n=1}^{\infty} b(n) \stackrel{\text{def}}{=} \lim_{N\to\infty} \prod_{n=1}^{N} b(n), \tag{B.3}$$

whenever this limit exists and has a finite, nonzero value.

Lemma B.13 (Weierstrass Product Test) *If the sequence $\{a(n) : n \in \mathbf{Z}^+\}$ is absolutely summable, then the infinite product of $\{1 + a(n) : n \in \mathbf{Z}^+\}$ exists and satisfies*

$$\prod_{n=1}^{\infty} |1 + a(n)| \leq \exp\left(\sum_{n=1}^{\infty} |a(n)|\right).$$

Proof: First note that $0 \leq \log(1 + |x|) \leq |x|$ for all real numbers x. Then, writing $b(n) = 1 + a(n)$, observe that $|b(n)| = |1 + a(n)| \leq 1 + |a(n)|$, and

$$\log\left(\prod_{n=1}^{N} |b(n)|\right) = \sum_{n=1}^{N} \log|b(n)| \leq \sum_{n=1}^{N} \log(1 + |a(n)|) \leq \sum_{n=1}^{\infty} |a(n)| < \infty.$$

This also gives the upper bound on the size of the product.

Since $\{a(n)\}$ is absolutely summable, its elements satisfy $|a(n)| < 1/2$ for all sufficiently large n. It may thus be assumed, extracting finitely many factors if necessary, that $a(n) > -1/2$ for all n. Notice that $\log|1 + x| \geq -(2\log 2)|x|$ for all $x > -1/2$. Thus we may write $\log|b(n)| = \log|1 + a(n)| \geq -(2\log 2)|a(n)|$, and observe that

$$\log\left(\prod_{n=1}^{N} |b(n)|\right) \geq -(2\log 2)\sum_{n=1}^{\infty} |a(n)| > -\infty.$$

This implies that $\prod_{n=1}^{\infty} |b(n)| > 0$. □

Change of variable formulas

Let $u = u(t)$ be a real-valued function on the closed and bounded interval $I = [a, b] \subset \mathbf{R}$, and suppose that its derivative u' exists and is continuous on I. Put $u(I) \stackrel{\text{def}}{=} \{u(t) : t \in I\} \subset \mathbf{R}$. Since u is continuous on I by Lemma B.7, $u(I)$ must be a closed and bounded interval by Corollary B.6.

If $f = f(x)$ is continuous on $u(I)$, then $(f \circ u)u'$ will be continuous on I and will satisfy the following:

$$\int_{u(a)}^{u(b)} f(x)\,dx = \int_{a}^{b} f(u(t))u'(t)\,dt. \tag{B.4}$$

This is Theorem 7.36 on page 164 of Apostol's *Mathematical Analysis*. It is not necessary for u to be an increasing function, so that $u(I) = [u(a), u(b)]$ and u gives a one-to-one correspondence between I and $u(I)$, since the factor u' will give negative weight to the right-hand integrand whenever u backtracks.

More generally, let $\mathbf{u} = \mathbf{u}(s,t) = (u(s,t), v(s,t))$ be a vector-valued function of two real variables, defined on the rectangular region $\mathbf{I} = \{(s,t) \in \mathbf{R}^2 : a < s < b, c < t < d\}$, where $-\infty \leq a < b \leq \infty$ and $-\infty \leq c < d \leq \infty$ so that $\mathbf{I}$ could be unbounded. Let $\mathbf{u}(\mathbf{I}) \stackrel{\text{def}}{=} \{\mathbf{u}(s,t) : (s,t) \in \mathbf{I}\} \subset \mathbf{R}^2$ be the possibly unbounded image region. Plane regions have more complicated geometry than intervals, so to generalize the change of variables formula we must make three assumptions on $\mathbf{u}$:

CV-1: We assume that $\mathbf{u}$ is one-to-one, namely $\mathbf{u}(s,t) = \mathbf{u}(s',t') \Rightarrow (s,t) = (s',t')$. Then points in the region $\mathbf{u}(\mathbf{I})$ are in one-to-one correspondence with points in $\mathbf{I}$. Thus we need not worry about backtracking or overlaps.

CV-2: We assume that all four partial derivatives $\frac{\partial u}{\partial s}$, $\frac{\partial u}{\partial t}$, $\frac{\partial v}{\partial s}$, and $\frac{\partial v}{\partial t}$ exist and are continuous at each point in $\mathbf{I}$. Then we may define the *Jacobian matrix* $\mathbf{u}'$ as follows:

$$\mathbf{u}'(s,t) \stackrel{\text{def}}{=} \begin{pmatrix} \frac{\partial u}{\partial s}(s,t) & \frac{\partial u}{\partial t}(s,t) \\ \frac{\partial v}{\partial s}(s,t) & \frac{\partial v}{\partial t}(s,t) \end{pmatrix}.$$

This matrix generalizes the derivative for vector-valued functions of several variables. It satisfies a condition similar to the one in Theorem B.8, namely

$$\mathbf{u}(s,t) = \mathbf{u}(s_0,t_0) + \mathbf{u}'(s_0,t_0)\begin{pmatrix} s-s_0 \\ t-t_0 \end{pmatrix} + o\left(|s-s_0| + |t-t_0|\right),$$

as $s \to s_0$ and $t \to t_0$.

CV-3: We assume that $J_{\mathbf{u}}(s,t) \neq 0$ for all $(s,t) \in \mathbf{I}$, where $J_{\mathbf{u}}$ is the determinant $\det \mathbf{u}'$ of the 2×2 matrix $\mathbf{u}'$. This is called the *Jacobian determinant* of $\mathbf{u}$, and is a continuous function on $\mathbf{I}$ defined by

$$J_{\mathbf{u}}(s,t) \stackrel{\text{def}}{=} \frac{\partial u}{\partial s}(s,t)\frac{\partial v}{\partial t}(s,t) - \frac{\partial v}{\partial s}(s,t)\frac{\partial u}{\partial t}(s,t).$$

This insures that no part of $\mathbf{u}(\mathbf{I})$ is pinched down to a curve or point.

Theorem B.14 *Suppose that* $\mathbf{u} = \mathbf{u}(s,t)$ *satisfies conditions CV-1, CV-2, and CV-3 on the open rectangular region* $\mathbf{I} = (a,b) \times (c,d) \in \mathbf{R}^2$. *Let* $f = f(x,y)$ *be a function continuous on* $\mathbf{I}$ *and suppose that* $\iint_{\mathbf{u}(\mathbf{I})} f(x,y)\,dxdy$ *exists. Then*

$$\iint_{\mathbf{u}(\mathbf{I})} f(x,y)\,dxdy = \int_{s=a}^{b}\int_{t=c}^{d} f(u(s,t),v(s,t))|J_{\mathbf{u}}(s,t)|\,dsdt. \qquad \text{(B.5)}$$

Proof: This is a special case of Theorem 15.11 combined with Theorem 15.7c of Apostol's *Mathematical Analysis.* □

For example, let $\mathbf{I} = (0,\infty) \times (0,2\pi) \subset \mathbf{R}^2$ and consider the change into polar coordinates $u(s,t) = s\cos t$, $v(s,t) = s\sin t$. Then for $\mathbf{u} = (u,v)$ we have $\mathbf{u}(\mathbf{I}) = \mathbf{R}^2 \setminus \mathbf{X}^+$, where $\mathbf{X}^+ = \{(x,0) : x > 0\}$ is the positive half of the x-axis in $\mathbf{R}^2$. The function $\mathbf{u}$ satisfies conditions CV-1, CV-2, and CV-3, since

$$\mathbf{u}'(s,t) = \begin{pmatrix} \cos t & -s\sin t \\ \sin t & s\cos t \end{pmatrix}; \qquad |J_{\mathbf{u}}(s,t)| = |s\cos^2 t + s\sin^2 t| = |s| = s \neq 0.$$

This change of variable provides a clever old way to show that $\int_{\mathbf{R}} e^{-\pi x^2}\,dx = 1$:

$$\begin{aligned} \left(\int_{-\infty}^{\infty} e^{-\pi x^2}\,dx\right)^2 &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-\pi x^2}e^{-\pi y^2}\,dxdy = \iint_{\mathbf{R}^2} e^{-\pi(x^2+y^2)}\,dxdy \\ &= \iint_{\mathbf{R}^2\setminus\mathbf{X}^+} e^{-\pi(x^2+y^2)}\,dxdy = \int_{s=0}^{\infty}\int_{t=0}^{2\pi} e^{-\pi s^2}\,s\,dsdt \\ &= \int_{s=0}^{\infty} e^{-\pi s^2}\,2\pi s\,ds = -e^{-\pi s^2}\Big|_{s=0}^{\infty} = 1. \qquad \text{(B.6)} \end{aligned}$$

The unsquared integral is $+1$ because the integrand $e^{-\pi x^2}$ is strictly positive.

Integrability

Riemann defined the integral as a limit of finite sums over finer and finer partitions of the interval $[a, b]$. It always exists for nice integrands and nice intervals:

Theorem B.15 *If $f = f(x)$ is continuous on a closed and bounded interval $I = [a, b] \subset \mathbf{R}$, then the Riemann interval $\int_a^b f(x)\,dx$ exists.*

Any complex-valued function $f = f(t)$ can be written as $f = \Re f + i\Im f$, where $\Re f = \Re f(t)$ and $\Im f = \Im f(t)$ are unique real-valued functions called, respectively, the *real part* and *imaginary part* of f. Here i satisfies $i^2 = -1$. Then f is continuous if and only if $\Im f$ and $\Re f$ are both continuous. If f is continuous on a closed and bounded interval $[a, b]$, it is Riemann integrable with

$$\int_a^b f(t)\,dt \stackrel{\text{def}}{=} \int_a^b \Re f(t)\,dt + i\int_a^b \Im f(t)\,dt.$$

Also, f is differentiable if and only if $\Im f$ and $\Re f$ are both differentiable, with $\frac{d}{dt}f(t) \stackrel{\text{def}}{=} \frac{d}{dt}\Re f(t) + i\frac{d}{dt}\Im f(t)$. Roughly speaking, for any property that is preserved by addition, f has the property if and only if both $\Re f$ and $\Im f$ have the property.

An *improper* Riemann integral of f is the limit of Riemann integrals as the interval of integration is enlarged to approach some discontinuity of f, or to fill out an unbounded domain. For example, the following is a typical improper Riemann integral:

$$\int_{-\infty}^0 e^x\,dx \stackrel{\text{def}}{=} \lim_{a\to\infty}\int_{-a}^0 e^x\,dx = \lim_{a\to\infty}\left[e^x\right]\Big|_{-a}^0 = \lim_{a\to\infty}\left(1 - e^{-a}\right) \quad = 1.$$

A function $f = f(t)$ is said to be *absolutely integrable* on an interval $[a, b]$ if $\int_a^b |f(t)|\,dt$ exists and is finite. This applies to real or complex functions f. If $a = -\infty$ or $b = +\infty$ or both, this definition means that a finite limit L exists such that for every $\epsilon > 0$, we can find finite endpoints a_0, b_0 with $a < a_0 < b_0 < b$ so that:

$$\left|\int_{a'}^{b'} |u(t)|\,dt - L\right| < \epsilon,$$

whenever $a < a' < a_0$ and $b_0 < b' < b$. A function $u = u(t)$ is said to be *square-integrable* if u^2 is absolutely integrable.

Lebesgue's definition of the integral, which is nicely discussed in Apostol's *Mathematical Analysis*, has the property that $\int_a^b |f(t)|\,dt$ exists and is finite if and only if $\int_a^b f(t)\,dt$ exists and is finite. In no case does cancellation of positive and negative parts of f affect whether the integral is computable; only the absolute size of f matters. This is not true for Riemann's integral: the function $f(t) = 1$ if t is rational, $f(t) = -1$ if t is irrational, which is not Riemann integrable on $[0, 1]$ because it has too many discontinuities, is nevertheless absolutely Riemann integrable on $[0, 1]$ because $|f(t)| = 1$ for all t so $|f|$ is continuous.

Differentiation of integrals

Suppose that $u = u_x(t)$ is a function of t, but also depends on a parameter x. Then $U(x) \stackrel{\text{def}}{=} \int_a^b u_x(t)\,dt$, where $[a,b]$ is some fixed interval, defines a function $U = U(x)$ at each value of x where u_x is continuous in t on all of $[a,b]$. This U inherits some of the properties of u_x:

Theorem B.16 *If, for all x in some open interval I, both $u_x(t)$ and $\frac{\partial}{\partial x}u_x(t)$ are continuous on $I \times [a,b]$, then the function $U(x) = \int_a^b u_x(t)\,dt$ exists and is differentiable at all $x \in I$, and the derivative may be expressed as a Riemann integral:*

$$U'(x) = \frac{d}{dx}U(x) = \int_a^b \frac{\partial}{\partial x}u_x(t)\,dt.$$

The function $U'(x)$ so defined is continuous at each $x \in I$. □

For example, consider $u_x(t) = \frac{1}{t}e^{-xt^2}$. Then $\frac{\partial}{\partial x}u_x(t) = -te^{-xt^2}$, so for any $0 < a < b < \infty$ and $x \neq 0$ we have:

$$U'(x) = \frac{d}{dx}\int_a^b \frac{e^{-xt^2}}{t}\,dt = \int_a^b -te^{-xt^2}\,dt = \frac{1}{2x}\left[e^{-xt^2}\right]\Big|_{t=a}^{b} = \frac{e^{-xb^2} - e^{-xa^2}}{2x}.$$

But $u_x(t)$ and $\frac{\partial}{\partial x}u_x(t)$ are both continuous at $x = 0$, so the function $U'(x)$ is continuous at $x = 0$ where its value is $\int_a^b(-t)\,dt = (a^2 - b^2)/2$. A little simplification gives the identity $\lim_{x\to 0}\frac{1}{x}(e^{-ax} - e^{-bx}) = b - a$.

B.5 Some basic probability theory

A *probability space* is a nonempty set X, a sufficiently large collection $\mathcal{A}$ of subsets $A \subset X$ called *events*, and a nonnegative *probability function* Pr defined at each $A \in \mathcal{A}$ and satisfying certain niceness conditions. Think of points in X as possible outcomes of an experiment or measurement, with a subset A being a range of outcomes. Then $\Pr(A)$ is the fraction of outcomes falling within the range A.

One such probability space is the discrete set $X = \{H, T\}$, the events $\emptyset$, $\{H\}$, $\{T\}$, and X itself, and the function Pr defined by $\Pr(\emptyset) = 0$, $\Pr(\{H\}) = \Pr(\{T\}) = \frac{1}{2}$, and $\Pr(X) = 1$. This is the familiar "fair coin toss," giving heads H or tails T with equal likelihood. More generally, a *discrete probability space* is any finite or countable set X.

$\mathcal{A}$ is sufficiently large in the following sense:

- X belongs to $\mathcal{A}$.
- If A belongs to $\mathcal{A}$, then so does A^c, the complement of A in X.
- For each sequence $\{A_i\}$ of subsets in $\mathcal{A}$, the union $\cup_i A_i$ belongs to $\mathcal{A}$.
- For each sequence $\{A_i\}$ of subsets in $\mathcal{A}$, the intersection $\cap_i A_i$ belongs to $\mathcal{A}$.

Such a collection of subsets is called a *sigma-algebra*, because it is closed under (countably infinite) algebra-like combinations such as union and intersection. The two-subset collection $\mathcal{A} = \{X, \emptyset\}$ is the simplest possible sigma-algebra.

Pr satisfies conditions derived from intuitive notions of probability:

1. $\Pr(X) = 1$, that is, X describes all outcomes under consideration.
2. $\Pr(\emptyset) = 0$, that is, a non-outcome need not be considered.
3. If $A, B \in \mathcal{A}$ and $A \subset B$, then $0 \leq \Pr(A) \leq \Pr(B) \leq 1$. That is, allowing more outcomes makes an event likelier.
4. $\Pr(\cup_i A_i) = \sum_i \Pr(A)$ for each sequence $\{A_i\}$ of *disjoint* subsets in $\mathcal{A}$. That is, the probabilities of nonoverlapping events simply add up.

For example, a set of $n > 1$ equally likely outcomes is described by the discrete probability space $X = \{1, 2, \ldots, n\}$, with $\mathcal{A}$ being all the 2^n different subsets of X and Pr defined by $\Pr(\{1\}) = \cdots = \Pr(\{n\}) = \frac{1}{n}$. From that and the fourth condition on Pr, we compute, for example, that $\Pr(\{1, 2\}) = 2/n$.

In general, to compute the probability of an event, decompose it into disjoint simple events whose probabilities are known, and then use the additivity of the probability function.

An example of a *continuum probability space* is the unit interval $X = [0, 1]$, with $\mathcal{A}$ being the sigma-algebra generated by all the open intervals (a, b), where $0 \leq a < b \leq 1$. These are often called the *Borel subsets* of $[0, 1]$. Pr is computed on the Borel sigma-algebra from conditions 1–4 and its values on intervals: $\Pr(a, b) \stackrel{\text{def}}{=} |b - a|$. More generally, for the same X and $\mathcal{A}$, we can specify another probability function by the formula

$$\Pr(a, b) \stackrel{\text{def}}{=} \int_a^b \rho(x)\, dx, \tag{B.7}$$

where $\rho = \rho(x)$ is a nonnegative and integrable *probability density function*, or *pdf*, satisfying $\Pr(0, 1) = \int_0^1 \rho(x)\, dx = 1$. Note that $\rho = \mathbf{1}$ gives the first example.

The real line $\mathbf{R}$ is another continuum probability space $\mathbf{R}$. Here we must use some nonconstant $\rho \geq 0$ with $\int_{\mathbf{R}} \rho(x)\, dx = 1$ to define the probability function on Borel subsets. The *Gaussian*, or *normal distribution*, is the probability function determined by the following pdf:

$$\rho(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{B.8}$$

where $\mu \in \mathbf{R}$ is called the *mean* and $\sigma > 0$ is called the *standard deviation*. These can be expressed as moment integrals of ρ, which are defined further on in Equation B.11. In that notation,[1] $\mu = M_1$ and $\sigma^2 = \mathrm{Var}(\rho) = M_2 - M_1^2$. Thus, if a probability function is known to be Gaussian, then it is completely determined by its first and second moments.

[1] We abuse notation by writing ρ instead of the unnamed probability function it determines.

The *cartesian product* of probability spaces can be defined, too. If X_1 and X_2 are probability spaces with respective sigma-algebras $\mathcal{A}_2, \mathcal{A}_2$ and probability functions P_1, P_2, then $X = X_1 \times X_2$ is another probability space. Its events $\mathcal{A}$ are the sigma-algebra generated by all products $A_1 \times A_2$, where $A_1 \in \mathcal{A}_1$ and $A_2 \in \mathcal{A}_2$. Its probability function is computed from its definition on such elementary product sets: $\Pr(A_1 \times A_2) \stackrel{\text{def}}{=} \Pr_1(A_1)\Pr_2(A_2)$. For example, rolling two fair dice gives a point in the cartesian product space $X_1 \times X_2$, where $X_1 = X_2 = \{1, 2, 3, 4, 5, 6\}$. Both factors are equipped with the sigma-algebras and probability functions of 6 equally likely outcomes. The event of rolling a total of 3 is a subset of the product space; it decomposes into the disjoint union of elementary product events $\{1\} \times \{2\} \cup \{2\} \times \{1\}$, giving a probability

$$\begin{aligned} \Pr(\{1\} \times \{2\} \cup \{2\} \times \{1\}) &= \Pr(\{1\} \times \{2\}) + \Pr(\{2\} \times \{1\}) \\ &= P_1(\{1\})P_2(\{2\}) + P_1(\{2\})P_2(\{1\}) \\ &= (\frac{1}{6})(\frac{1}{6}) + (\frac{1}{6})(\frac{1}{6}) = \frac{1}{18}. \end{aligned}$$

Various counting principles and algebraic techniques are used to find such a decomposition so that a probability can be computed. For example, the probability that the sum of the dice is an odd number can be computed as

$$p_{odd} = \sum_{x_1 + x_2 \text{ odd}} \Pr(x_1)\Pr(x_2).$$

This calculation can be avoided. Observe that an odd sum can be obtained in the same number of ways as an even sum: just swap the odd and even numbers on one of the dice. Thus $p_{even} = p_{odd}$, with p_{even} defined in the obvious way. Since $p_{even} + p_{odd} = 1$, we conclude that $p_{odd} = \frac{1}{2}$.

The product construction works similarly for $N > 2$ spaces $X_1, \ldots, X_N$. Repetitions of a measurement can often be modeled by such a product probability space with all factors the same, and then they are called *Bernoulli trials*, with each measurement described as *independent* and *identically distributed.*

A *random variable* is a real-valued function f on a probability space X which is nice enough so that $\{x \in X : f(x) > y\}$ belongs to the events collection $\mathcal{A}$, for every real value y. It is thereby possible to compute the probability that f takes values in some interval, because the outcomes leading to that value form a set for which Pr is defined. For example, the sum of two fair dice is a random variable on the product probability space $X = \{1, \ldots, 6\} \times \{1, \ldots, 6\}$. We may write $f(x_1, x_2) = x_1 + x_2$ in the coordinates (x_1, x_2) of that space. In another example, if $X \subset \mathbf{R}$, then $f(x) = x$ can serve as a random variable. *Random vectors* are vector-valued function on X nice enough so that each component of f is a random variable. For example, if X is the unit square in $\mathbf{R}^2$, then $f(x_1, x_2) = (x_1 + x_2, x_1 - x_2)$ is a random vector.

The *expected value* of a random variable f is its average on all of X, computed using the probability function. In the discrete case, X can be enumerated and $\mathcal{A}$

contains all single-point subsets of X:

$$E(f) = \sum_{x \in X} f(x)\Pr(\{x\}). \tag{B.9}$$

In the case $X = [0,1] \subset \mathbf{R}$ with the Borel sigma-algebra $\mathcal{A}$, we must approximate single-point sets using open subintervals from $\mathcal{A}$. This is similar to integration with Riemann sums: fix a disjoint sequence $\{A_i\}$ of subsets from $\mathcal{A}$ such that $X = \cup_i A_i$, then pick $x_i \in A_i$ and approximate $E(f) \approx \sum_i f(x_i)\Pr(A_i)$. If f is a continuous function, this will tend to the Riemann integral $\int_0^1 f(x)\rho(x)\,dx$ as the decomposition of X is refined, namely as $\max_i \Pr(A_i) \to 0$. We may therefore define the expected value of such a random variable on a continuum probability space X to be the integral

$$E(f) = \int_{x \in X} f(x)\rho(x)\,dx. \tag{B.10}$$

Note the similarities between Equations B.10 and B.9.

An idea of Lebesgue can sometimes be used to overcome the technical problem of evaluating the integral: Let $R(y) = \Pr(\{x : f(x) > y\})$ be the *cumulative distribution function* defined by f. Then

$$E(f) = \int_{-\infty}^{+\infty} y\,dR(y),$$

whenever this integral exists. Note that R decreases from 1 to 0 as y goes from $-\infty$ to $+\infty$.

The *moments* of a probability function are the expected values of the basic random variables $1, x, x^2, \ldots$:

$$M_k = E(x^k) = \begin{cases} \displaystyle\sum_{x \in X} x^k \Pr(\{x\}), & \text{if } X \text{ is a discrete probability space;} \\ \displaystyle\int_{x \in X} x^k \rho(x)\,dx, & \text{if } X \text{ is a continuum.} \end{cases} \tag{B.11}$$

Since both sums and integrals preserve finite sums, we can compute the expected value of any polynomial random variable $f(x) = a_0 + a_1 x + \cdots + a_n x^n$ from these pieces: $E(f) = a_0 M_0 + \cdots + a_n M_n$. Note that $M_0 = 1$ in all cases.

The first moment M_1 of a probability function Pr, if it exists, is called its *mean*. If in addition the second moment M_2 exists, then $\mathrm{Var}(\Pr) \stackrel{\text{def}}{=} M_2 - M_1^2$ is called its *variance*. By adding and subtracting and using the definitions, we can show

$$M_2 - M_1^2 = E(x^2) - 2M_1 E(x) + M_1^2 = E(x^2 - 2M_1 x + M_1^2) = E([x - M_1]^2).$$

This is the more familiar definition of variance: $\mathrm{Var}(\Pr) = E([x - E(x)]^2)$ is the expected value of squared deviation from the mean. This calculation also proves that $\mathrm{Var}(\Pr) \geq 0$, or that $M_2 \geq M_1^2$ whenever both moments exist.

B.6 Some more advanced results

We sketch the proofs of some useful results that are unfortunately beyond the scope of this text. While important technical issues must be left unresolved at our level of analysis, I believe the outlines provided here are useful guides for further study.

Theorem B.17 (Plancherel) *Suppose that $u = u(x)$ and $v = v(x)$ belong to $L^2(\mathbf{R})$, and have Fourier integral transforms $\hat{u}$ and $\hat{v}$, respectively. Then $\langle \hat{u}, \hat{v} \rangle = \langle u, v \rangle$, namely,*

$$\int_{\xi=-\infty}^{\infty} \overline{\hat{u}(\xi)}\, \hat{v}(\xi)\, d\xi = \int_{x=-\infty}^{\infty} \overline{u(x)}\, v(x)\, dx.$$

Proof: For small $\epsilon > 0$, let ϕ_ϵ be the function defined by

$$\phi(\xi) = e^{-\pi\xi^2}; \qquad \phi_\epsilon(\xi) \stackrel{\text{def}}{=} \phi(\epsilon\xi) = e^{-\pi(\epsilon\xi)^2}.$$

We use ϕ and ϕ_ϵ because we know their Fourier integral transforms:

$$\hat{\phi}(x) = \phi(x); \qquad \widehat{\phi_\epsilon}(x) = \frac{1}{\epsilon}\phi(\frac{x}{\epsilon}).$$

Now, ϕ_ϵ is bounded by 1, continuous and rapidly decreasing, hence integrable. Also, for each $\xi \in \mathbf{R}$, $\phi_\epsilon(\xi) \to 1$ as $\epsilon \to 0$. Thus, $\phi_\epsilon w$ is an integrable function whenever $w = w(\xi)$ is integrable, and

$$\lim_{\epsilon\to 0} \int_{\mathbf{R}} \phi_\epsilon(\xi) w(\xi)\, d\xi = \int_{\mathbf{R}} [\lim_{\epsilon\to 0} \phi_\epsilon(\xi)] w(\xi)\, d\xi = \int_{\mathbf{R}} w(\xi)\, d\xi.$$

On the other hand, if $g = g(x, y)$ is integrable on $\mathbf{R}^2$, then

$$\lim_{\epsilon\to 0} \int\int_{\mathbf{R}^2} g(x,y) \frac{1}{\epsilon}\phi(\frac{y-x}{\epsilon})\, dxdy = \int_{\mathbf{R}} g(x,x)\, dx.$$

Both of these limits may be evaluated using the Lebesgue Dominated Convergence Theorem (Theorem 10.27 on page 270 in Apostol's *Mathematical Analysis*).

Now let $w(\xi) = \overline{\hat{u}(\xi)}\, \hat{v}(\xi)$, apply the first limit identity, expand the two Fourier transforms as integrals, and interchange the order of integration:

$$\begin{aligned}
\langle \hat{u}, \hat{v} \rangle &= \lim_{\epsilon\to 0} \int_{\xi\in\mathbf{R}} \phi_\epsilon(\xi) \overline{\hat{u}(\xi)}\, \hat{v}(\xi)\, d\xi \\
&= \lim_{\epsilon\to 0} \int\int\int_{\mathbf{R}^3} \phi_\epsilon(\xi) \overline{e^{-2\pi i x\xi} u(x)} e^{-2\pi i y\xi} v(y)\, d\xi dx dy \\
&= \lim_{\epsilon\to 0} \int\int_{\mathbf{R}^2} \overline{u(x)} v(y) \left[\int_{\mathbf{R}} \phi_\epsilon(\xi) e^{2\pi i(x-y)\xi}\, d\xi \right] dxdy \\
&= \lim_{\epsilon\to 0} \int\int_{\mathbf{R}^2} \overline{u(x)} v(y) \frac{1}{\epsilon}\phi(\frac{y-x}{\epsilon})\, dxdy \;=\; \int_{\mathbf{R}} \overline{u(x)} v(x)\, dx \;=\; \langle u, v \rangle .
\end{aligned}$$

For the last step, we let $g(x, y) = \overline{u(x)} v(y)$ and used the second limit identity. □

Hilbert spaces

A *Hilbert space* $\mathbf{X}$ is an inner product space that supports approximation by being *complete*: if an infinite sequence $\{\mathbf{u}_n\} \subset \mathbf{X}$ settles down as $\mathbf{n} \to \infty$, in the sense that for each $\epsilon > 0$ there is a sufficiently large N such that $\|\mathbf{u}_n - \mathbf{u}_m\| < \epsilon$ for all $n, m \geq N$, then it defines a vector $\mathbf{u}_\infty = \lim_{n\to\infty} \mathbf{u}_n$ that is also in $\mathbf{X}$. Sequences that settle down this way are called *Cauchy sequences*.

All finite-dimensional Euclidean spaces, whether real or complex, are complete (See Apostol's *Mathematical Analysis*, Theorem 4.8, page 73) and thus are Hilbert spaces. The space $\ell^2(\mathbf{N})$ is also complete (See Royden's *Real Analysis*, Problem 12, page 126). However, the inner product space **Lip** is not complete. Let $\mathbf{u}_n$ be defined by $u_n(t) = t^n$ for $n = 1, 2, \ldots$, all of which are evidently Lipschitz continuous on $[0, 1]$. Then

$$\|\mathbf{u}_n\|^2 = \int_0^1 (t^n)^2 \, dt = \frac{1}{2n+1},$$

so $\|\mathbf{u}_n - \mathbf{u}_m\| \leq \|\mathbf{u}_n\| + \|\mathbf{u}_m\| = \frac{1}{\sqrt{2n+1}} + \frac{1}{\sqrt{2m+1}}$, which is at most $\frac{2}{\sqrt{2N+1}}$ for any $n, m \geq N$. Thus $\{\mathbf{u}_n : n = 1, 2, \ldots\}$ is a Cauchy sequence. However, $u_n(1) = 1$ for all n, whereas $u_n(t) \to 0$ for all $0 \leq t < 1$, so the limit function $\mathbf{u}_\infty$ is discontinuous at $t = 1$ and cannot belong to **Lip**.

Completing an inner product space $\mathbf{X}$ to get a Hilbert space means:

1. Appending to the set $\mathbf{X}$ all limits of Cauchy sequences in the original $\mathbf{X}$;

2. Identifying two elements $\mathbf{u}, \mathbf{v}$ of the augmented $\mathbf{X}$ as equal if $\|\mathbf{u} - \mathbf{v}\| = 0$.

It is necessary to extend the inner product to the completed set by using limits and to take care of other technicalities. For example, we must show that limits of Cauchy sequences in the augmented $\mathbf{X}$ are already limits of Cauchy sequences in the original $\mathbf{X}$. Also, individual elements of the completed inner product space can be obtained by way of more than one infinite sequence in $\mathbf{X}$, so we must devise a sensible convention for labeling them. These problems arise already for the real line, which is the completion of the rational numbers: for instance, $1 = 0.999\ldots$ has at least two decimal representations.

An inner product space can always be completed to a Hilbert space. The exercises in Royden, pages 146–147, explore the details of this in great generality.

Subsets closed under linear combination are considered subspaces of a Hilbert space only if they are complete. Finite-dimensional linear spans are complete and hence are subspaces, but it is also possible to define the (complete) subspace generated by infinitely many vectors of an infinite-dimensional Hilbert spaces. For example, $\overline{\text{span}}\{\mathbf{v}_i : i \in \mathbf{N}\}$ is defined to be the completion of the subspace consisting of all vectors of the form $\sum_{i=0}^{m} a_i \mathbf{v}_i$, for any m and any scalars $a_0, a_1, \ldots, a_m$. Another example is the sequences in ℓ^2 that have zeroes at all prime indices; these form a complete, infinite-dimensional subspace of ℓ^2.

Finite-dimensional Hilbert spaces have orthonormal bases by the Gram-Schmidt construction, Theorem 2.7. The generalization of such a basis to an infinite dimen-

sional Hilbert space $\mathbf{X}$ requires some completeness property. So, an infinite subset $\mathbf{B} = \{\mathbf{b}_n : n \in \mathbf{N}\} \subset \mathbf{X}$ is called a *Hilbert basis* if it satisfies the following:

Hilbert Basis Axioms

Orthogonality: If $n \neq m$, then $\langle \mathbf{b}_n, \mathbf{b}_m \rangle = 0$;

Normalization: For all n, $\langle \mathbf{b}_n, \mathbf{b}_n \rangle = 1$;

Completeness: If $\langle \mathbf{v}, \mathbf{b}_n \rangle = 0$ for all n, then $\mathbf{v} = 0$.

For example, $\{\mathbf{e}_n : n \in \mathbf{N}\}$ is an Hilbert basis for $\ell^2(\mathbf{N})$. It is sometimes more convenient to use $\mathbf{Z}$ or $\mathbf{Z}^+$ for the index set. An example from Fourier analysis, studied in Chapter 3, is the orthonormal basis $\{\mathbf{b}_n : n \in \mathbf{Z}\}$ for **Lip** and L^2 defined by the complex exponential functions $b_n(t) \stackrel{\text{def}}{=} e^{2\pi i n t}$.

One way to get subspaces of a Hilbert space is by orthogonal projection:

Theorem B.18 *If $\mathbf{X}$ is a Hilbert space and $P : \mathbf{X} \to \mathbf{X}$ is an orthogonal projection, then $P\mathbf{X}$ is a Hilbert subspace with the same norm and inner product as $\mathbf{X}$.*

Proof: We must show that $P\mathbf{X}$ is complete. But if $\{\mathbf{x}_k\}$ is a Cauchy sequence in $P\mathbf{X}$, it is a Cauchy sequence in $\mathbf{X}$ so it has a limit $\lim_{n\to\infty} \mathbf{x}_k = \mathbf{x}_\infty \in \mathbf{X}$. It is only necessary to show that $\mathbf{x}_\infty \in P\mathbf{X}$, which is true if $\mathbf{x}_\infty = P\mathbf{x}_\infty$. But $\mathbf{x}_k = P\mathbf{x}_k$, so $\|\mathbf{x}_k - P\mathbf{x}_\infty\| = \|P\mathbf{x}_k - P\mathbf{x}_\infty\| = \|P(\mathbf{x}_k - \mathbf{x}_\infty)\| \leq \|\mathbf{x}_k - \mathbf{x}_\infty\|$, which means that $\lim_{k\to\infty} \mathbf{x}_k = P\mathbf{x}_\infty = \mathbf{x}_\infty$. □

Given any orthonormal set of vectors $\mathbf{B} = \{\mathbf{b}_n\} \subset \mathbf{X}$, finite or infinite, we may form the orthogonal projection

$$P_{\mathbf{B}} : \mathbf{X} \to \mathbf{X}; \qquad P_{\mathbf{B}}\mathbf{x} \stackrel{\text{def}}{=} \sum_n \langle \mathbf{b}_n, \mathbf{x} \rangle \, \mathbf{b}_n.$$

We may then show that the range of $P_{\mathbf{B}}$ is exactly $\overline{\text{span}}\,\mathbf{B}$, the completion of the linear span of the vectors in $\mathbf{B}$.

B.7 Rising cut-off functions

We finish by applying the results in this appendix to the characterization of all rising cutoff functions that may be used for fraying and splicing.

Notice that any function satisfying Equation 3.11 must be of the form

$$r(t) \stackrel{\text{def}}{=} e^{i\rho(t)} \sin\theta(t), \tag{B.12}$$

where ρ and θ are real-valued functions satisfying

$$\rho(t) = \begin{cases} 2n\pi, & \text{if } t < -1, \\ 2m\pi, & \text{if } t > 1; \end{cases} \qquad \theta(t) = \begin{cases} 0, & \text{if } t < -1, \\ \frac{\pi}{2}, & \text{if } t > 1; \end{cases} \qquad \theta(t) + \theta(-t) = \frac{\pi}{2}. \tag{B.13}$$

Of course, any real-valued rising cut-off function can have $\rho \equiv 0$. In any case, $|r(-t)| = |\sin[\frac{\pi}{2} - \theta(t)]| = |\cos\theta(t)|$, so $|r(t)|^2 + |r(-t)|^2 = 1$.

Let $r = r(t)$ be any differentiable rising cut-off function. Then $\frac{d}{dt}|r(t)|^2$ is a symmetric continuous bump function supported on $[-1, 1]$. This observation provides a mechanism for parameterizing all rising cut-offs. Start with an integrable bump function $\phi = \phi(t)$ satisfying the following conditions:

$$\phi(t) = 0 \text{ if } |t| > 1; \quad \phi(t) = \phi(-t) \text{ for all } t; \quad \int_{-\infty}^{\infty} \phi(s)\, ds = \frac{\pi}{2}. \tag{B.14}$$

Then obtain $\theta(t) = \int_{-1}^{t} \phi(s)\, ds$. But any integrable function ϕ defined on $[0, 1]$ with nonzero integral can be multiplied by some constant and extended by reflection about zero to satisfy Equation B.14.

As an application of this construction, we start with a quadratic polynomial and obtain a cubic spline for the angle function:

$$\phi(t) = \frac{3\pi}{8}(1-t)(1+t); \qquad \theta(t) = \frac{\pi}{8}\left[2 + 3t - t^3\right]; \qquad -1 \le t \le 1. \tag{B.15}$$

The example cut-off r_0 of Equation 3.12 comes from $\phi(t) = \frac{\pi}{4}\mathbf{1}_{[-1,1]}$ and $\theta(t) = \frac{\pi}{4}(1+t)$ for $t \in [-1, 1]$. Likewise, taking $\phi(t) = \frac{\pi^2}{8}\cos\frac{\pi}{2}t$ for $t \in [-1, 1]$ yields r_1 from $\theta(t) = \frac{\pi}{4}(1 + \sin\frac{\pi}{2}t)$.

B.8 Further Reading

- Tom M. Apostol. *Calculus*, volume I. John Wiley & Sons, New York, second edition, 1967. ISBN 0-471-00005-1.
- Tom M. Apostol. *Calculus*, volume II. John Wiley & Sons, New York, second edition, 1969. ISBN 0-471-00007-8.
- Mark Davis, Michael Everson, Asmus Freytag and John H. Jenkins. *Unicode 3.2 Code Charts.* Unicode Technical Committee, 2002. Available from URL `http://www.unicode.org`.
- Gene H. Golub and James M. Ortega. *Scientific Computing and Differential Equations.* Academic Press, San Diego, 1992. ISBN 0-12-289255-0.
- Steven G. Krantz. *Techniques of Problem Solving.* American Mathematical Society, Providence, Rhode Island, 1997. ISBN 0-8218-0619-X.
- Halsey L. Royden. *Real Analysis.* Macmillan Publishing Company, 866 Third Avenue, New York, third edition, 1988. ISBN 0-02-404151-3.
- Allen B. Tucker. *Programming Languages.* McGraw–Hill, New York, 1986. ISBN 0-07-065416-6.
- Mladen Victor Wickerhauser. *Adapted Wavelet Analysis from Theory to Software.* A K Peters, Ltd., Natick, Massachusetts, 1994. ISBN 1-56881-041-5.

This Page Intentionally Left Blank

Index

absolutely integrable, 287
absolutely summable, 283
accuracy, 124
adjoint, 47
    matrix, 47
    periodic, 160
Aff, affine group, 136
algorithm, 1, 275
algorithmic complexity, 179
alphabet, 180
antisymmetry, HA or WA, 169
ASCII, 179, 273
assignment, 276

band-limited function, 120
bandwidth, 120
basis, 28
    Schauder, 32
Bessel's inequality, 66
binary tree, 184
binomial coefficient, 76
biorthogonal
    dual, 40
    filters, 169
bit rate, 188
`bits()`, 9
Borel subsets, 289

canonical Huffman tree, 196
cas, 98
casting out nines, 205
`castout9()`, 206
`castout11()`, 206
`castout255()`, 208
`castout257()`, 208
Cauchy sequence, 13, 293
Cauchy–Schwarz inequality, 37
character, 180
Chebyshev
    nodes, 113
    polynomial, 112
`chebyshevpoly()`, 114
checksum, 205
code, 181
    Huffman, 193
    linear, 201
    prefix, 182
    uniquely decipherable, 181
compact support, 73
complete, 293
completeness, 152
    periodic, 176
complexity, 179
composition, 45
conditional probability density, 124
conditionally summable, 283
conditionals, 276
conditioning, 18
conjugacy, 152
containment, 147
continuous, 281
continuum probability space, 289
control characters, 273
conventional indexing, 152
CQF, 151
CRC-32, 218
CSG (Checksum Goals), 207

d-ary tree, 185
decimals, 9
decision function, 127
decrease, 147

decrements, 276
`decipherex()`, 181
degree, 108
dense, 32
dependency relation, 27
derivative, 281
`det()`, 62
determinant, 55, 62
DFT, 95
DHT, 98
`digits()`, 9
dilation, 147
    operator, 85
dimension, 28
discrete probability space, 288
discrete wavelet decomposition, 155
dist (Hamming), 203
`divdiff()`, 111
divided difference, 110
division algorithm, 2
divisor, 3
doubly infinite sequence, 283
DWT, 145
`dwt()`, 160
`dynamicdecoding()`, 264
`dynamicencoding()`, 219

E, E^+, 197
efficient coding, 188
encoding, 181
entropy, 187
`epwconstant()`, 115
`epwlinear()`, 130
error
    representation, 18
    round-off, 14
    truncation, 14
estimator, 127
    unbiased, 128
Euclid, 12
Euclid's algorithm, 4
    extended, 8
    for mod-2 polynomials, 269
    mod-2 polynomials, 213
events, 288
expected value, 290

factorization, 5
faithful, 137
father function, 148
FFT, 95, 98
`fft()`, 98
FHT, 100
`fht()`, 100
filter, 146
filter length, 152
filter transform, 157
    periodic, 160
finite algorithm, 275
finite impulse response (FIR), 146
finite-energy signals, 35
float, 15
`float.h`, 15
floats, 1
floor, 2
focused, 124
for loop, 277
Fourier coefficients, 83
Fourier series, 83
    exponential, 86
`fray()`, 80
fraying, 79

Gaussian, 289
    elimination, 57
    noise, 126
    sinc function, 122
Gaussian function, 93
`gcd()`, 4
`gcdx()`, 8
`gepp()`, 58
`gilbertcode()`, 268
Givens rotation, 57
graph, 183
greatest common divisor, 3

Haar filter, 152
`haarlift()`, 166
Haar measure, 140
half-sample symmetry, 168

Hamming distance, 203
hexadecimal, 9
Hilbert basis, 294
Hilbert–Schmidt norm, 49
Horner's method, 108
`horner()`, 108
host, 10
`hsldht()`, 262
`hslhaarfilter()`, 261
`hslidht()`, 262
`hslihaarfilter()`, 261
`hslpredict()`, 261
`hslupdate()`, 261
`hsp()`, 168
Huffman code, 193

ideal value, 123
iFFT, 98
`ifft()`, 98
`ihaarlift()`, 167
`ildht0()`, 167
ill-conditioned, 19
imprecision, 123
inaccuracy, 124
increase, 147
increments, 276
independence, 152
    periodic, 176
infinite decimal expansions, 13
infinite product, 284
infinite sequence, 282
information source, 180
inputs, 275
instructions, 275
integers, 1
    negative, 2
    positive, 2
integrable
    absolutely, 287
    square, 287
interior vertex, 185
Intermediate Value Theorem, 281
interpolation condition, 117
`intmod2polydegree()`, 212
`intmod2polydivision()`, 213
`intmod2polygcd()`, 269
`intmod2polyirreducibe()`, 214
`intmod2polyproduct()`, 211
`intmod2polysum()`, 210
inverse Fourier transform, 88
`ipcqfilter()`, 162
`ipdwt()`, 257
`ipdwt0()`, 163

Jacobian, 286
joint probability density, 124

Kolmogorov complexity, 179
Kronecker delta, 38, 149

L^2, 34
ℓ^2, 33
Lagrange interpolation, 109
`lagrange()`, 110
`ldht0()`, 167
Lebesgue integral, 287
left invariant integral, 140
left invariant measure, 140
left sided limit, 280
Leibniz' rule, 76
length preserving, 56
letter, 180
lifting, 166
limit, 280, 283
`limits.h`, 10
linear
    code, 201
    combination, 26
    independence, 27
    span, 27
    transformation, 42
Lip, 34
Lipschitz condition, 34, 282
`lnormalize()`, 172
loop, 82
    for, 277
    while, 277
lower triangular, 56
LU decomposition, 59
`lupp()`, 60

Mallat algorithm, 157
mantissa, 13
matrix, 42
    coefficients, 44
    representation, 136
mean, 291
Mean Value Theorem, 282
measurement density, 124
memory, 275
message, 180
    encoding function, 181
    instance, 180
meta-characters, 273
Minkowski inequality, 37
`mipp()`, 59
mod-2 polynomial, 209
`mod2polychecksum()`, 217
`mod2polydegree()`, 211
`mod2polydivision()`, 212
`mod2polygcd()`, 213
`mod2polyint()`, 210
`mod2polyproduct()`, 211
`mod2polysum()`, 210
moments, 291
mother function, 133
MRA, 147
multipliers, 276
multiresolution analysis, 147

N, natural numbers, 2
Newton form, 110
`newtonpoly()`, 111
noise, 126
nondegeneracy, 29
norm axioms, 29
normal density, 126
normal distribution, 289
normalization, 151
    periodic, 176
normalizers, 276

O, 197
occurrence probability, 180
`onormalize()`, 261
operator norm, 49
orthogonal, 24
    basis, 38
    complement, 38
    CQF, 151
    MRA, 148
orthonormal basis, 38
outputs, 275
overflow, 12

parity, 197
Parseval's formula, 67, 84
partial sums, 283
`pcqfilter()`, 162
pdf, 289
`pdwt()`, 163
`pdwt0()`, 163
%, remainder operator, 2
periodic, 70
    extension, 75
    filter transform, 160
periodization, 70
    smooth local, 82
permutation matrix, 63
piecewise function, 115
Plancherel's theorem, 292
pointwise convergence, 283
polarization, 26
Poly, 33
polynomial, 108
power spectrum, 85
precision, 123
prediction, 171
prefix code, 182
prime, 5
    factorization, 5
    mod-2 polynomials, 214
probability density function, 289
probability space, 288
projection, 51
pseudocode, 275
`pwconstant()`, 115
`pwlinear()`, 116
pyramid algorithm, 157

Q, rational numbers, 12

quadrature filter, 146
quantization, 123
quasi-inverse, 6
quaternary, or quadtree, 185

R, real numbers, 13
`r1()`, 78
random variable, 290
rapid decrease, 284
relatively prime, 5
    mod-2 polynomials, 214
remainder operator (%), 2
representation, 137
return, 277
Riemann integral, 287
right invariant integral, 251
right invariant measure, 174
right sided limit, 280
rigid motion, 56
rising cut-off function, 78, 294
rotation, 56
round-off error, 14

sampling theorem, 121
scalars, 26
scale invariance, 117
scaling function, 145, 148
Schauder basis, 32
    unconditional, 33
scientific notation, 13
self-orthonormality, 151
    periodic, 176
selfadjoint, 48
sequence, 282
side-effect, 277
sign-and-magnitude, 11
signal to noise ratio, 126
sinc function, 92
    Gaussian, 122
size preserving, 162
smooth, 76
source, 180
span, 27
spectral density, 120
`splice()`, 80
splicing, 79
spreading in support, 159
sqrt, 277
square-integrable, 287
square-summable, 283
`sslu()`, 61
sublinearity, 29
submultiplicativity, 49
subspace, 27
summable, 283
support, 73
symmetric functions, 117
symmetry, HS or WS, 168

trace, 50
transformation, 42
translation, 56
tree
    binary, 184
    canonical Huffman, 196
    d-ary, 185
    Huffman, 193
    quaternary, 185
truncation error, 14
two-scale relation, 146
twos complement, 11

unconditional basis, 33
underflow, 12
Unicode, 275
uniform convergence, 283
unique factorization, 5
uniquely decipherable code, 181
unitary, 56
    representation, 139
    transformation, 138
updating, 171
upper triangular, 56

Vandermonde matrix, 95
variance, 291
vector, 26
vertex, 185
Vol (Hamming sphere), 203

wavelet expansion, 157

wavelet transform, 140
    inverse, 143
weighted depth, 191
well-conditioned, 18
while loop, 277
white noise, 126
whole-sample symmetry, 168
wide characters, 275
`wsl42dwt()`, 173
`wsl42filter()`, 172, 173
`wsl42idwt()`, 173
`wsl42ifilter()`, 172
`wsl97dwt()`, 260
`wsl97idwt()`, 260
`wsl97ifilter()`, 260
`wslpredict()`, 171
`wslupdate()`, 171
`wsp()`, 169

Z, integers, 2
zero-one matrix, 63